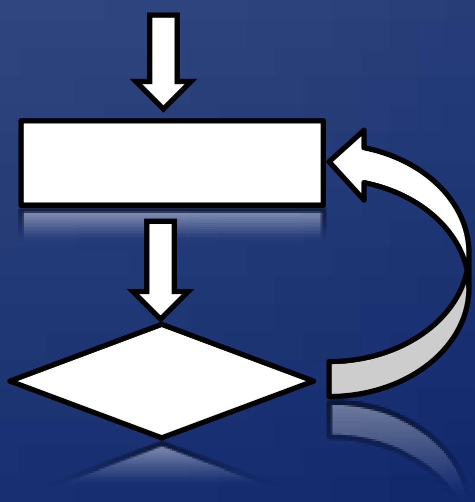
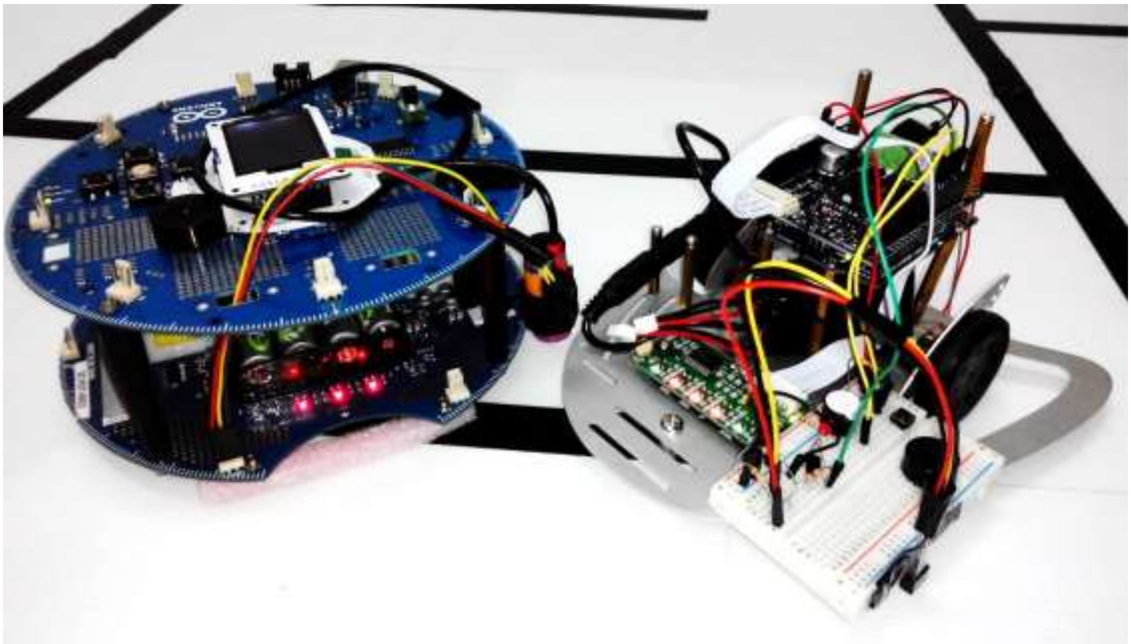
Faculty of Engineering and Information Technology
Southern University College

BUILD A LINE FOLLOWER ROBOT

A User-Friendly Guide

Edited by
Chia Kim Seng
Kang Eng Siew
Eng Pei Chee
Hui Seng Keong

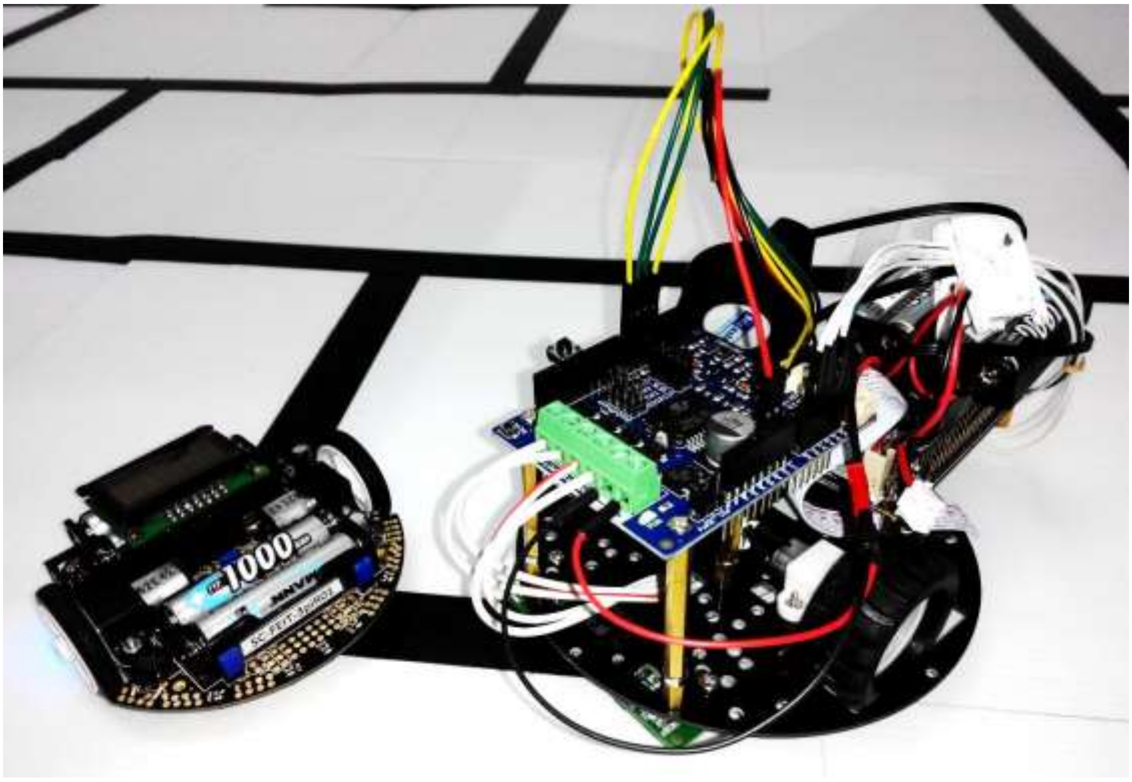




SOUTHERN UNIVERSITY COLLEGE

BUILD A LINE FOLLOWER ROBOT

A USER-FRIENDLY GUIDE



EDITED BY: CHIA KIM SENG, KANG ENG SIEW, ENG PEI CHEE, HUI SENG KEONG

Build a Line Follower Robot

A User-Friendly Guide

Edited by

Chia Kim Seng

Kang Eng Siew

Eng Pei Chee

Hui Seng Keong

Published by: Southern University College
PTD 64888, 15KM, Jalan Skudai,
P.O.Box 76, 81300 Skudai, Johor, Malaysia
Tel: + 607 558 6605
Fax: + 607 556 3306
Website: http://www.southern.edu.my

Sponsored by: Southern University College Research Fund (SUCRF)
Institute of Graduate Studies and Research (IGSR),
Southern University College

Perpustakaan Negara Malaysia Cataloguing-in-Publication Data

BUILD A LINE FOLLOWER ROBOT : A User-Friendly Guide /
Edited by Chia Kim Seng, Kang Eng Siew, Eng Pei Chee, Hui Seng Keong
Also available in print
Reproduction of: BUILD A LINE FOLLOWER ROBOT : A User-Friendly Guide. Skudai, Johor : Kolej Universiti Selatan, 2014
Mode of access: World Wide Web
Title from PDF cover
eISBN 978-983-2453-46-8
1. Robotics. 2. Robots--Design and construction. 3. Electronic books.
I. Chia, Kim Seng, 1986-. II. Kang, Eng Siew, 1985-.
III. Eng, Pei Chee, 1984-. IV. Hui, Seng Keong, 1970-.
629.892

Contents

Preface

Applying the knowledge and skills that students gained from their formal learning to solve “real-life” engineering problems plays a crucial role to enhance overall teaching and learning process in higher education system. There is a need to find a better way to equip the students with not only a broader spectrum of knowledge, but also reasoning skills that will allow them to work in a wide range of industries.

We believe that practical works improve students’ academic interest and performance in engineering courses. This belief motivates us to complete this handbook to guide learners to practice their theoretical knowledge using practical assignment by means of a line follower robot in engineering courses.

The main scope of this handbook is to address the practical concerns of building a line follower robot. A line follower robot is a mobile machine that is designed to move along a given line. This kind of robot has been widely implemented for various purposes e.g. transporting goods in manufacturing industries, research experiments, and competitions. Thus, the development of a line follower robot provides an alternative practical opportunity for instructors and learners to enrich teaching and learning experiences.

This handbook aims to assist readers to build a line follower robot using an open-source electronics platform based on easy-to-use hardware and software – Arduino. In addition to the use of Arduino IDE, this handbook also presents the use of Atmel software and FPGA to program and to control the robot, respectively.

The content of this handbook is categorized into two parts, i.e. Getting Started and Examples of Implementation. In Getting Started, the procedure to getting started with 3pi robot using Arduino IDE (Chapter 1) and Atmel 6 (Chapter 2) are presented. Chapter 3 and Chapter 4 introduce Arduino robot and FPGA, respectively. Besides, we also provide the procedures of self-deploying line follower robots using Arduino UNO (Chapter 5) and FPGA (Chapter 6) as controllers.

In Part 2, our results and findings are presented. These results and findings are compiled into three topics of Robot Racing (Chapter 7), Maze Puzzle (Chapter 8), and C Programming Learning (Chapter 9).

Chia Kim Seng
August 2014

Acknowledgments

We express our sincere appreciation to the editors who have contributed their invaluable knowledge and experience to enrich the content of this handbook. Besides that, we would like to gratefully acknowledge the Institute of Graduate Studies and Research (IGSR) and the Faculty of Engineering and Information Technology of the Southern University College for providing the Southern University College Research Fund (SUCRF) and excellent environment for completing our line follower robots and this handbook. Last but not least, we would like to acknowledge the constructive advices and supports provided by the dean of IGSR, Prof. Dr Ho Khai Leong.

Thank you.

Authors
August 2014

Editors

Chia Kim Seng received his BEng and PhD in electrical and electronic engineering from Universiti Teknologi Malaysia, in 2010 and 2014, respectively. His team and he represented Malaysia to participate Asia-Pacific Broadcasting Union (ABU) ROBOCON 2007 in Vietnam and were awarded the title of Second Runner-up. His main research interest is in the field of Machine Learning, Artificial Intelligence, Control, Microcontroller, and Soft Modeling. He has served as a peer reviewer of numerous international journals e.g. RCS Analytical Method and IEEE sensor.

Kang Eng Siew received her BEng and PhD in electrical and electronic engineering from the Universiti Teknologi Malaysia in 2008 and 2013, respectively. After that, she joined the postdoctoral program under the Computational Nanoelectronics Research group, Universiti Teknologi Malaysia and has been active in research related to carbon based devices. Her main research interest is in the emerging area of nanoelctronics devices, focusing on the use of carbon based materials. She is presently with the Southern University College as an assistant professor.

Eng Pei Chee received her M.Eng degree in Electrical Engineering and B.Eng degree in Electrical and Electronics Engineering, from Universiti Tecknologi Malaysia. From year 2007-2008 she worked as a project engineer with the V.S. Electronic Sdn. Bhd. She joined the Southern University College in 2011 as a lecturer in the Department of Electrical & Electronic Engineering. Her research interests include field-programmable gate array (FPGA), VLSI technology, computer architecture, and biometrics.

Hui Seng Kheong received his MSc in Signal Processing from Lancaster University (UK) in 1996 and Ph.D in Electrical and Electronic from Universiti Teknologi Malaysia(UTM) in 2003. From 2003-2007, he worked as a manager with THScan, a biotechnology company that specializing thermal texture maps (TTMs) and scanning machines. He joined Southern College (later upgraded as Southern University College) in 2007 as Deputy Dean of Faculty of Engineering and Technology. His research interests are Engineering Mathematics, networking, communications, and signal processing.

Contributors

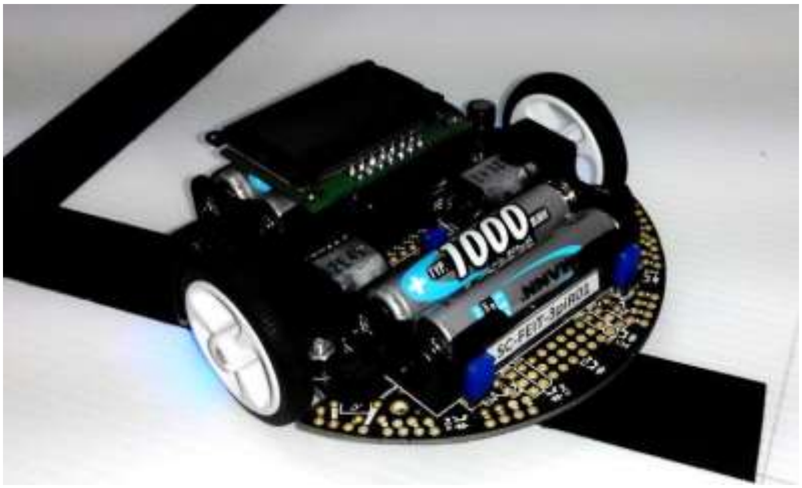
Chia Kim Seng
Faculty of Engineering and Information Technology
Southern University College
Skudai, Johor
Malaysia
respect_exist@hotmail.com

Eng Pei Chee
Department of Engineering and Electronic Engineering
Faculty of Electronic and Information Technology
Southern University College
Skudai, Johor
Malaysia
pceng@sc.edu.my

Kang Eng Siew
Department of Engineering and Electronic Engineering
Faculty of Electronic and Information Technology
Southern University College
Skudai, Johor
Malaysia
eskang@sc.edu.my

Chapter 1 Getting Started with 3pi Robot: Arduino IDE

Chia Kim Seng, Kang Eng Siew, Eng Pei Chee



1.1 Introduction

Orangutan programmer is the default programmer for 3pi robot. This chapter aims to provide essential procedures for first time users to getting started their project with 3pi robot quickly. The procedures of relevant software installation, driver installation, and sample programs are included. These procedures have been tested using Microsoft Windows 7 and Microsoft Windows 8.

1.2 Software and driver installation

The procedures that involved to getting started with 3pi robot using Orangutan programmer coupled with Arduino IDE are stated as follows:

1. **Install Arduino IDE in a computer.**
   - The latest Arduino Environment can be found at http://arduino.cc/en/Main/Software.
   - On the webpage (Figure 1.1), click on the compatible download link to download the latest Arduino integrated development environment (IDE) software.

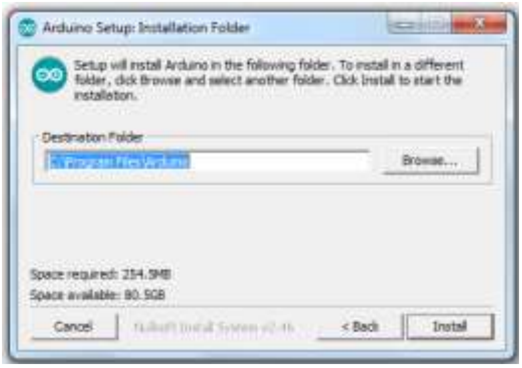
Figure 1.1 For Windows users, double-clicking on *Windows Installer*

- Select ***Save File***.

- Open the containing folder and run it.
- Review the software licensing agreement and agree to its terms by clicking ***I Agree*.**

- In the **Installation Options** dialog box, select all of the components for installation, and then click **NEXT.**

- Specify the name of the folder where you want to install Arduino products. You can either accept the default installation folder or click **Browse** to select the desired one.

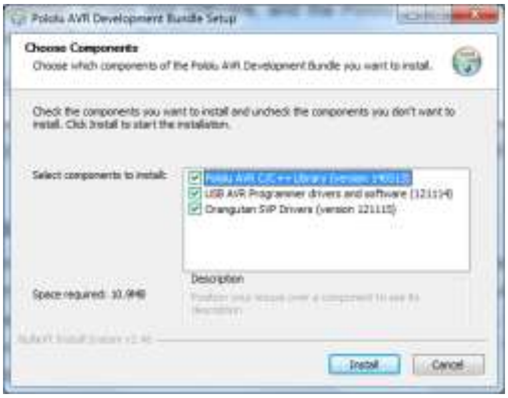


- When it is installing, the installer displays the following dialog box.

- Once the installation is completed, click **Close.**

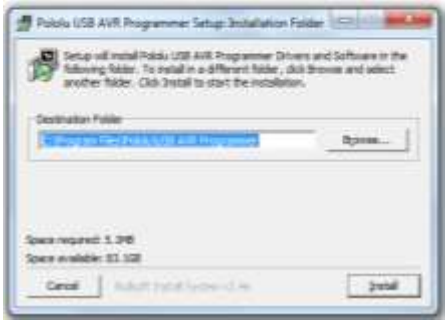
2. **Install Orangutan Driver to the computer.**
   - Download the latest drivers of "***Pololu AVR Development Bundle***". By July 2014, the latest is "… ***Windows (release 140513)***" that was available at http://www.pololu.com/file/0J541/pololu-avr-bundle-140513.exe or http://www.pololu.com/product/1300/resources.

   - Open the containing folder, and then run the installer.
   - In the **Pololu AVR Development Bundle Setup** dialog box, select all of the components for installation, and then click **INSTALL** to proceed with the installation.

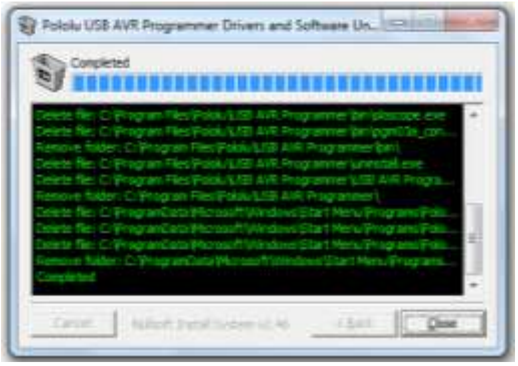


Note: Pololu AVR C/C++ Library is only required for Atmel Studio IDE (Chapter 2).

- During the installation, accept the default installation folder to install **Pololu USB Programmer Drivers** and **Pololu Orangutan SVP Drivers** by clicking **Install**.



- Once the installation is completed, click **Close** to quit the dialog box.



3. **Modify Arduino IDE for 3pi Pololu robots**.
   - Two files of **boards.txt** and **programmers.txt** in the directory of *C:/…/arduino/hardware/arduino/* (normally it can be found in the sub-folder of Program Files (x86)) is required to be replaced by the **modified** boards.txt and programmers.txt, respectively, using copy-and-paste manner. Please require the files from the authors.
   - Alternatively, users can modify these two files via pasting the code that stated in Appendix A to the end of the text files accordingly using administrator's authority.
   - If the modification is successful, the option to choose Pololu Orangutan as the desired **Board** and AVR ISP v2 as the desired **Programmer** should be available in the Arduino IDE.

4. **Install library for Polulo 3pi robot**.

- Pololu Arduino libraries files can be found at http://www.pololu.com/docs/0J17/5 (last access: August 2014).

- In the webpage, double-clicking on **Arduino libraries for the Orangutan and 3pi Robot** to open the download page (if applicable).

- Download the latest Pololu Arduino Libraries (**zip file**). If you are using the old version of Arduino IDE, you need to update the IDE with the latest version.

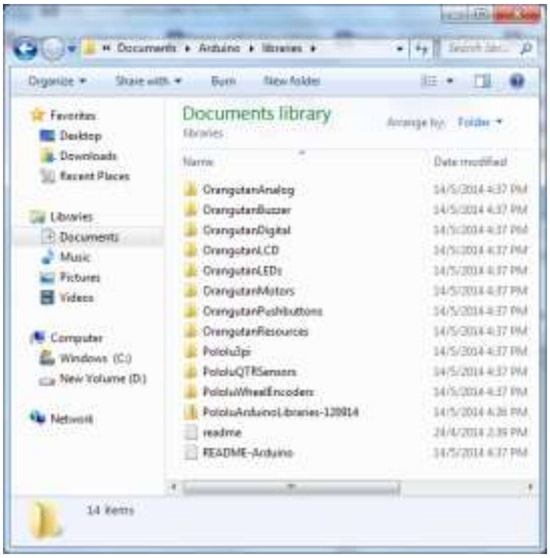
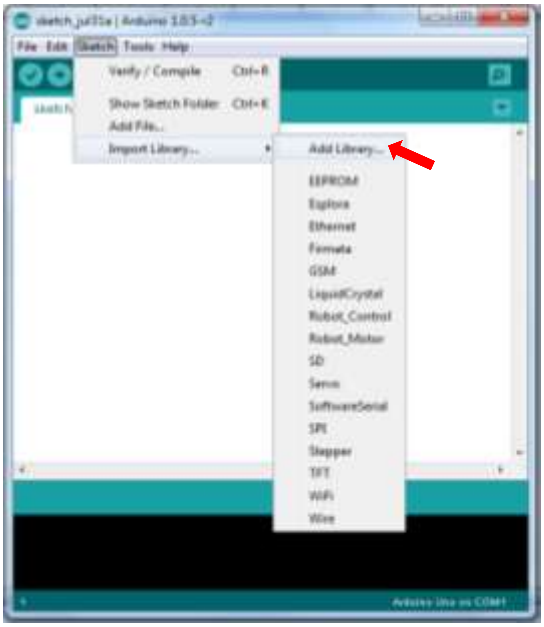


Figure 1.2 The directory that stores related libraries for 3pi robot

- After unzipping/extracting the zip file, we need to relocate all the library files to the directory of “My Documents/Arduino/ libraries/” or “Documents/Arduino/ libraries/” as that illustrated in Figure 1.2.

- If those libraries cannot be found in the Arduino IDE (please check at **File->Examples** and **File->Sketchbook->libraries** in the Arduino IDE), you may need to manually import the libraries by selecting **Sketch-> Import Library...-> Add Library...** in the Arduino IDE, and then browse the library .zip file.



1.3 Connect Orangutan programmer to Arduino IDE

This section summarises the important procedure that ensures the communication between a computer and a 3pi robot using Orangutan programmer can work properly.

1. Connect Orangutan programmer to a computer using its USB port. For Windows 8 users, the Windows will automatically install required driver when the programmer is plug-in to the computer.

2. Open Arduino IDE and then change the settings of the IDE as follows.

   a. **Board menu:** *Tools -> Board menu -> Pololu Orangutan*

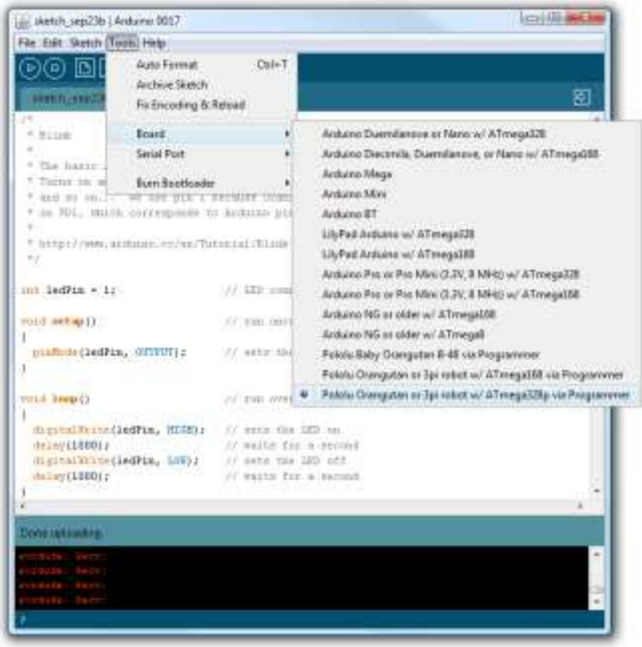


Figure 1.3 Arduino IDE - Board menu

   b. **Programmer:** *Tools -> Programmer -> AVR ISP v2*

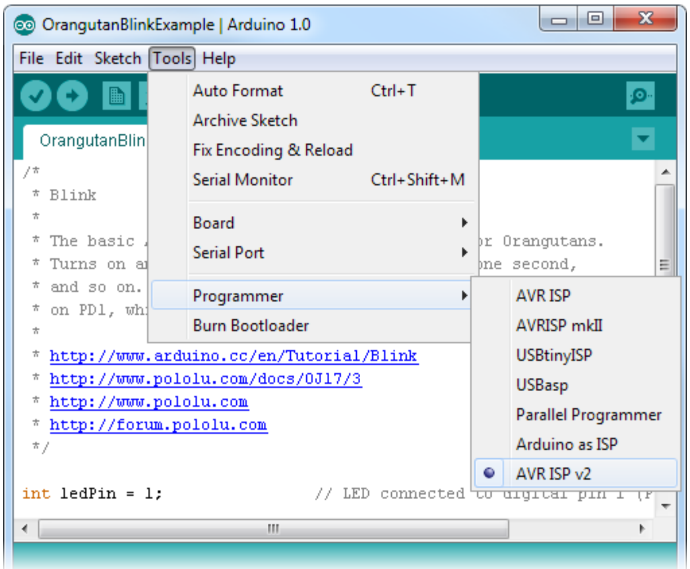


Figure 1.4 Arduino IDE – Programmer

c. Programmer's **serial port**: *Tools -> Serial Port ->COM*

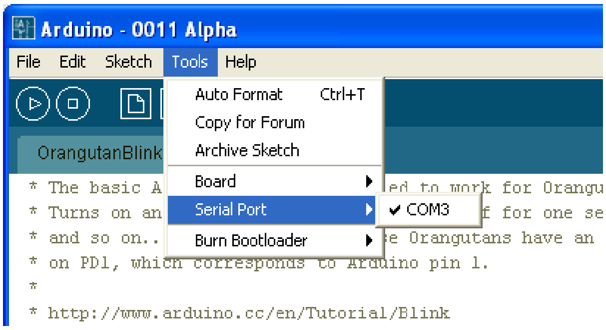


Figure 1.5 Arduino IDE - Serial Port

Note: The correct Serial Port is dependent on the USB port that has been connected to 3pi robot. Thus, in order to select the correct Serial Port, for Windows users, please go to Device Manager as that illustrated below.

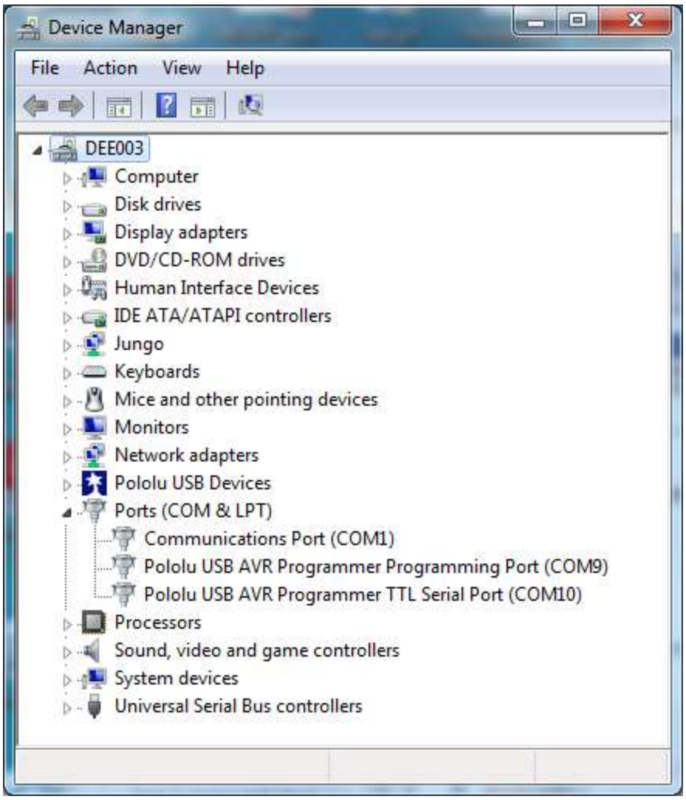


Figure 1.6 Arduino IDE - Serial Port: Device Manager indicates that the correct Serial Port is COM 9 (Programming Port)

1.4 Simple programs

After completing the first two steps (Section 1.2 and Section 1.3), we can upload a simple program (such as LED blink) to a 3pi robot. Nonetheless, we need to follow the following procedure and precautions to protect the robot.

1. Connect Orangutan programmer to a 3pi robot (six pins socket) properly.
2. **Turn on** the robot **before** uploading program into the robot.
3. **Ensure that the battery is sufficient.** Intuitively, the voltage should **more than 4.6V** while a set of four full charged AAA batteries should supply more than 5V to the robot. Insufficient battery can damage the robot during uploading. This can be done using *read_battery_millivolts* function as shown in Section 1.4.1.

1.4.1 Read battery voltage

"read_battery_millivolts() - performs 10 bits analog-to-digital conversions on the battery voltage sensing circuit of the 3pi robot and returns the average result in millivolts. A value of 5234 implies that the voltage of the batteries is 5.234 V. For four fully charged AAA rechargeable NiMH batteries, the voltage usually starts at a value above 5 V, and then **drops to around 4 V before the robot shuts off**. Therefore, monitoring this number is helpful for determining when we should charge the batteries."

(Reference: Pololu AVR Library Command Reference)

```
#include <PololuQTRSensors.h>
#include <OrangutanAnalog.h>
#include <OrangutanLCD.h>

 void setup()
{


}

 void loop()
{
     int bat = read_battery_millivolts();
     clear();
     print_long(bat);
     print("mV");
     lcd_goto_xy(0,1);
     delay(100);
}
```

Figure 1.7 Arduino IDE - Serial Port

1.4.2 LED blinking

According to the Pololu 3pi robot simplified schematic diagram, the digital pin No. 1 and No. 7 are connected to LED. Thus, the code that stated in Figure 1.8 can be used to demonstrate a simple LED blinking.

```
LEDBlink
int ledR = 1;
int ledG = 7;

 void setup()
{
  pinMode(ledR, OUTPUT);
  pinMode(ledG, OUTPUT);
}

 void loop()
{
  digitalWrite(ledR, HIGH);
  digitalWrite(ledG, HIGH);
  delay(200);
  digitalWrite(ledR, LOW);
  digitalWrite(ledG, LOW);
  delay(200);
}
```

Figure 1.8 Arduino IDE – LED Blinking

First, **ledR** and **ledG** are defined as pin 1 and pin 7, respectively. After that, these two pins were defined as outputs in **setup** function using **pinMode** function. Lastly, **digitalWrite** function was used to ON or OFF the LED using HIGH to indicate ON, and vice versa. **delay** function was used to increase the duration of the state (ON or OFF) of the LED. This means that the lower the value of the delay function, the faster the LED blinks.

Tips:

Please **verify** the source code using Arduino IDE before uploading it to a 3pi robot. This practice helps us to figure out the source problem when an error appears.

Chapter 2 Getting Started with 3pi Robot: Atmel 6

Eng Pei Chee

2.1 Introduction

This chapter provides a tutorial on how to get started programming the Atmel AVR microcontroller on a 3pi robot in Windows using the Atmel Studio 6 IDE.

Note:
This user guide is demonstrated and tested using Windows Vista, Windows 7 and Windows 8. For Windows XP users, users may refer to http://www.pololu.com/docs/0J51 for the installation.

2.2 Safety precautions before programming

Before programming, make sure your device is powered. If you are using rechargeable batteries for your device, make sure they are fully charged. Never attempt to program your device if the batteries are drained or uncharged. Losing power during programming could permanently disable a 3pi robot. It is always a good practice to include the battery voltage monitoring function (See Section 1.4.1) provided in the library in your program to ensure sufficient amount of voltage supplied to the 3pi robot.

2.3 Installation

2.3.1 Installing Atmel Studio 6

Download and install Atmel Studio 6 (current version is 6.2 with size of approximately 520 MB, updated May 2014), an integrated development environment (IDE) from Atmel.

The download link of Atmel Studio 6.2 Installer is available at
http://www.atmel.com/tools/atmelstudio.aspx?tab=overview.

2.3.2 Installing the Pololu AVR Development Bundle

After installing Atmel Studio, download and install **Pololu AVR Development Bundle**. The download link of **Pololu AVR Development Bundle** is available at
http://www.pololu.com/file-redirect/avr-development-bundle.

When you are prompted to choose which components should be installed during the installation process, select **ALL** the three components as that illustrated in Figure 2.1.

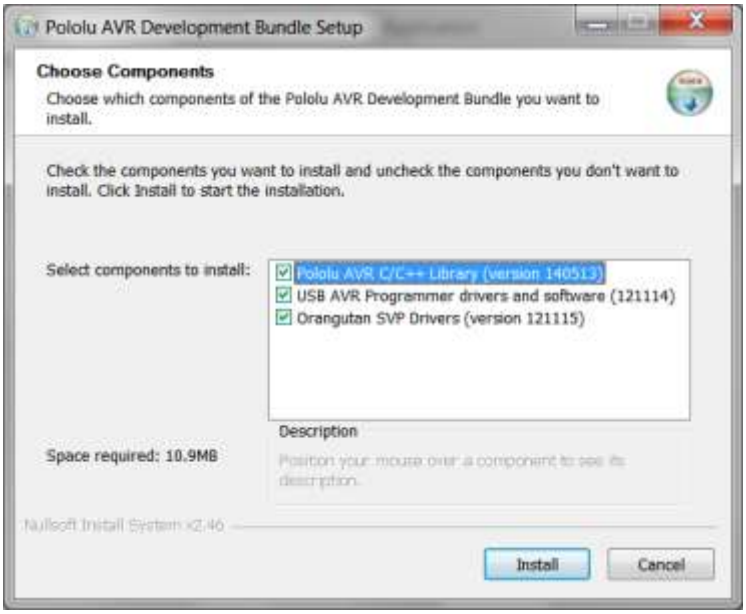


Figure 2.1 The screen of the Pololu AVR C/C++ Library installer

The descriptions of the three components are as follows:

i. **Pololu AVR C/C++ Library** - This library is required for Pololu Products i.e. the Orangutan robot controllers and 3pi robot. [Reference: User's guide @ http://www.pololu.com/docs/0J20]
ii. **Pololu USB AVR Programmer** – This library contains Pololu USB AVR Programmer drivers, configuration utility, and SLO-scope software. [Reference: User's guide @ http://www.pololu.com/docs/0J36]

iii. **Orangutan SVP drivers** – This is the robot **motor controller**'s driver that designed for Atmel AVR microcontroller module or ATmega1284P [Reference: User's guide @ http://www.pololu.com/docs/0J39]

Select "**Atmel Studio 6.2**" when you are prompted to "Install the library into the following toolchains:". If the Atmel Studio 6.2 checkbox is grayed out, then the installer is unable to find Atmel Studio 6.2. This suggests that you are required to reinstall or repair the installer.

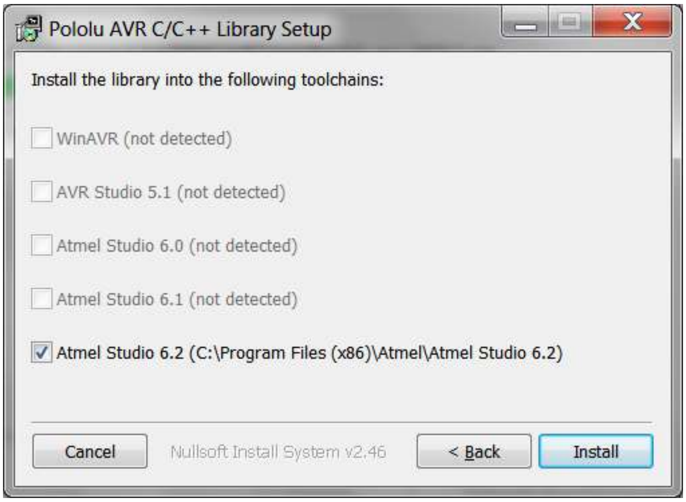


Figure 2.2 A screenshot to associate Pololu AVR C/C++ library with Atmel Studio 6.2.

During the installation, clicking **Install** when Windows asks you to install the drivers.

2.4 Setting up AVR Programmer

Follow the following procedures to configure Atmel Studio 6.2 to use the programmer.

2.4.1 Configure Pololu USB AVR Programmer.

Plug the Pololu USB AVR Programmer into your computer's USB port. Your computer should automatically install the necessary drivers.

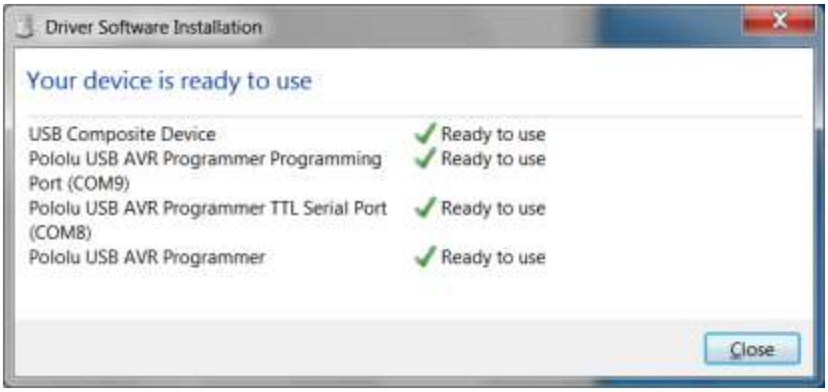


Figure 2.3 Configuring Pololu USB AVR Programmer, Drivers Installation

After installing the drivers, go to your computer's Device Manager and expand the "Ports (COM & LPT)" list. You should see two COM ports:

"Pololu USB AVR Programmer Programming Port"

and

"Pololu USB AVR Programmer TTL Serial Port".

In parentheses after these names, you will see the name of the port (e.g. "COM9" or "COM8").

Additionally, there should be a "Pololu USB Devices" list with an entry for the programmer.

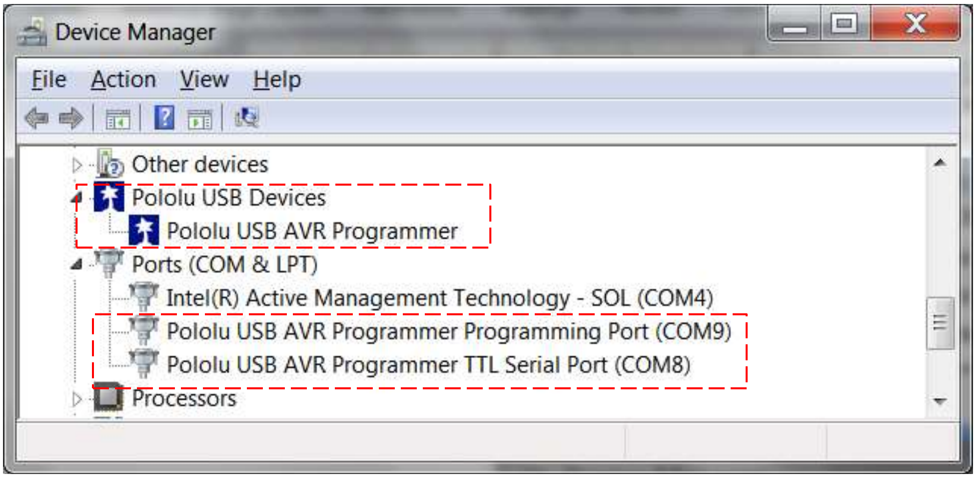


Figure 2.4 Device Manager: The Pololu USB AVR Programmer is recognized

2.4.2 Configure Atmel Studio to use the programmer.

Open Atmel Studio, with the programmer connected to your computer via USB, and then select **Add target…** from the **Tools** menu.

Apply the following settings, and then click **Apply**:

i. For the tool, select **STK500**.

ii. For the serial port, select the **COM port** that has been assigned to the programmer's programming port.

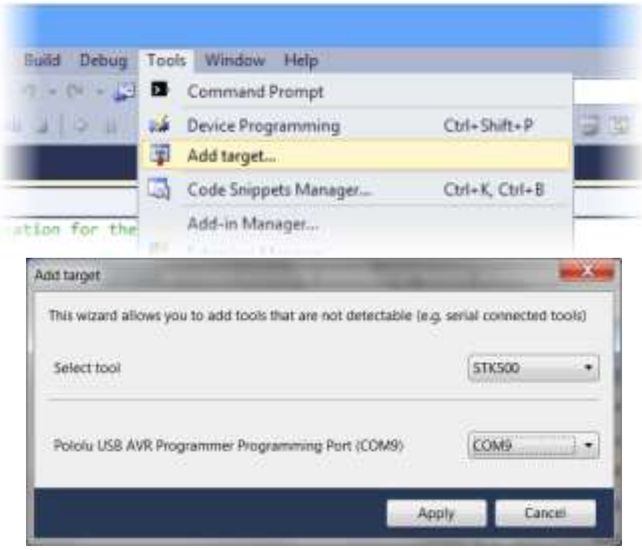


Figure 2.5 The **Add target** dialog box in Atmel Studio.

2.5 Compiling a simple program

2.5.1 Creating a new project:

Open Atmel Studio 6.2, and then create a new project by selecting **New -> Project…** in the File menu.

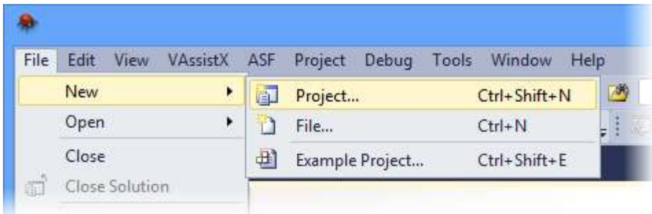


Figure 2.6: The **New** menu in Atmel Studio 6.

In the **New Project** dialog, select **Installed Templates** -> **C/C++** -> **Pololu**. Select the template **3pi Robot with ATmega328p**, and click **OK**.

Remark: If there is no Pololu category, make sure that you have installed the Pololu AVR C/C++ Library and then try to restart Atmel Studio.

Figure 2.7 The New Project dialog in Atmel Studio 6.

A new project from the template will be opened up as shown in below Figure.

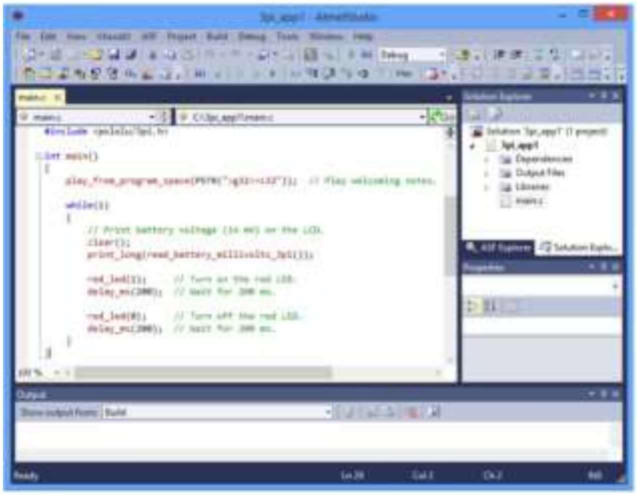


Figure 2.8 Installed template program

The template contains some simple codes that demonstrate basic features of the board, i.e. playing some notes on the buzzer, displaying character on LCD, and LED blinking.

The functions that being called are defined in the Pololu AVR C/C++ Library [http://www.pololu.com/docs/0J20].

2.5.2 Build the project

Build/compile the project by selecting **Build -> Build Solution** or by pressing **F7**. If the build is successful, then the build output will be shown in the “Output” tab at the bottom should end like this:

```
...
Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

2.5.3 Program the AVR (3pi Robot)

Before you start programming the AVR, make sure your 3pi robot is turned on. **Losing power during programming could permanently disable your 3pi.**

You can program by selecting either **Continue** or **Start Without Debugging** from the **Debug** menu or simply press the **F5** key.

If the programming is successful, you should be able to see the program running on your device.

- The buzzer should play a note whenever the device starts up.
- Battery voltage should be displayed on the LCD.
- The red user LED on the device should be blinking.

Chapter 3 Getting started with Arduino Robot

Chia Kim Seng

3.1 Introduction

Arduino Robot (http://arduino.cc/en/Main/Robot) consists of a build-in programmer that is used to communicate with a computer using a Universal Serial Bus (USB) A-to-B cable. This chapter aims to provide essential procedure for first time users to getting started their project with Arduino Robot quickly. The procedure includes relevant software installation, driver installation and a sample program demonstration.

3.2 Software and driver installation

The procedure involved to getting started with Arduino Robot is as follows:

i. **Install Arduino IDE in a computer.** The latest Arduino Environment can be found at http://arduino.cc/en/Main/Software.

ii. **Connect the Arduino Robot to a computer, and then install the required driver**. The details can be found at http://arduino.cc/en/Guide/Robot.

3.3 Connect Arduino IDE to Arduino Robot

In this section, the following procedure is required to ensure the communication can work correctly.

i. Connect Arduino Robot to the computer using USB A-B cable.

ii. **Open Arduino IDE** and **then change the settings of the IDE** as follows.

  a. Board menu: **Tools > Board > Arduino Robot Control**

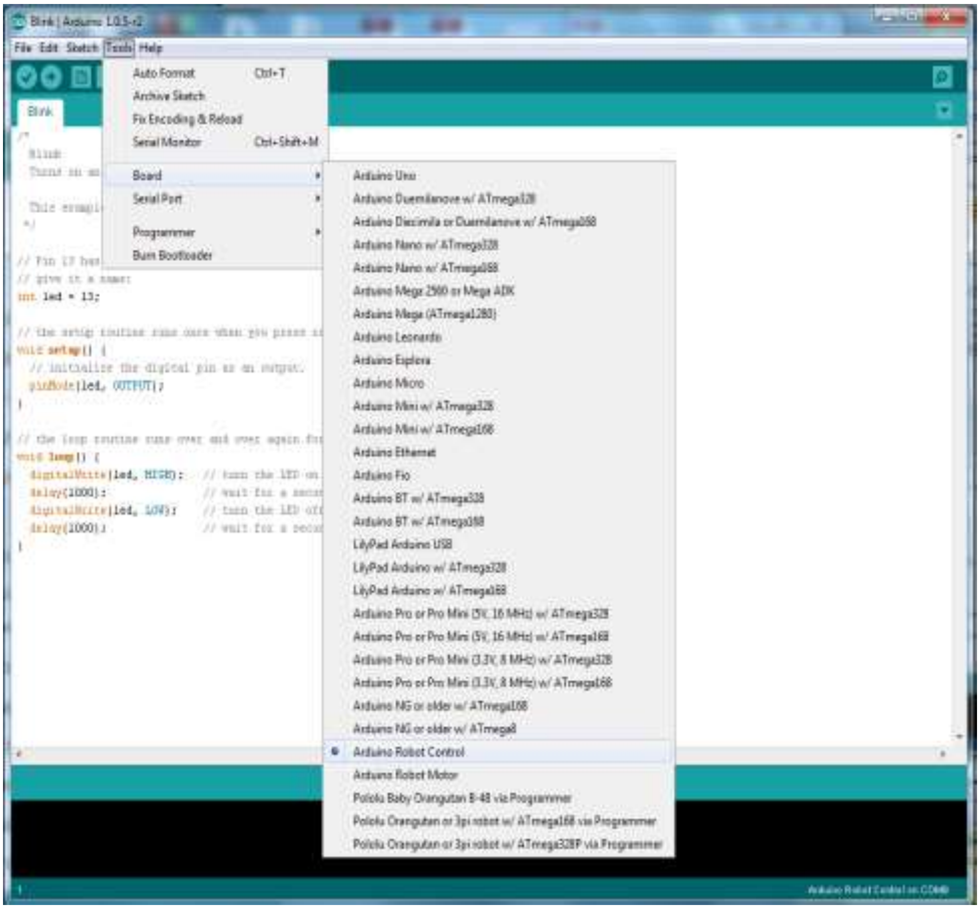


Figure 3.1 Arduino IDE - Board menu

b. Programmer: **Tools > Programmer > Arduino as ISP**

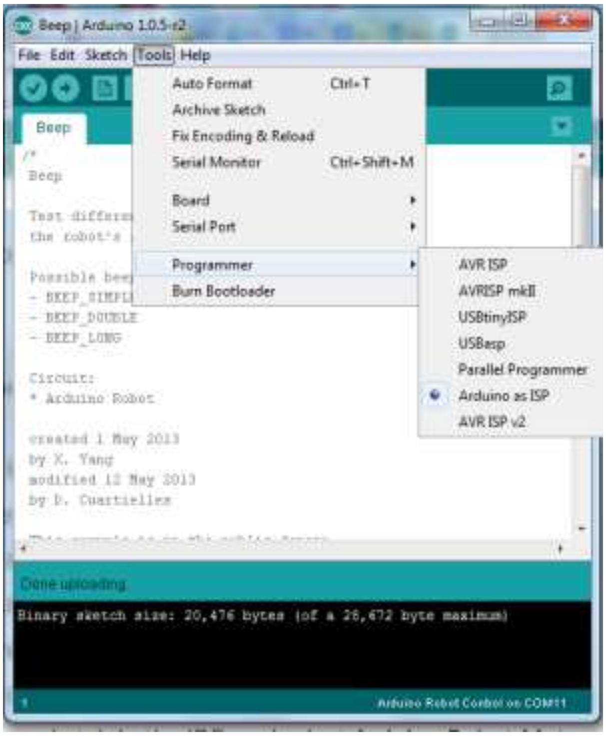


Figure 3.2 Arduino IDE – set Programmer - Arduino as ISP

c. Programmer's serial port: **Tools > Serial Port**

Figure 3.3 Arduino IDE - Serial Port

Note: The correct Serial Port is dependent on the USB port that has been connected to Arduino robot.

3.4 Discussion

The following sub-sections discuss the operations and the challenges of Arduino Robot, as well as proposed solution upon the challenges.

3.4.1 Control Board Vs Motor Board

Arduino robot consists of two separate boards, i.e. **Control Board** (top) and **Motor Board** (bottom). The general function of **Control Board** is to send a command or communicate with **Motor Board** so that a given task e.g. line following will be done by the **Motor Board**. **Motor Board**, on the other hand, can either follow the command given by **Control Board** or even feedback the command upon the completion of the given task. In other words, the main "job scopes" of the **Motor Board** are to control the speed and the direction of the motors, and to receive the signals of the Infrared (IR) sensors.

3.4.2 Line Following – Challenge

The Control Board should not be used to directly analyse the signals from IR sensors and instruct the Arduino Robot to follow a line. This is because the extensive updates of the signals of the IR sensors and also the speed of motor via sending commands from the Control Board to Motor Board and vice versa can cause significant lagging problems.

Therefore, we should only use Control Board to send commands e.g. "follow a line" and "chase an object" instead of a routine instruction. The Motor Board, on the other hand, should execute a given command accordingly without any further instruction e.g. the speed and direction of motors. This practice is similar to an effective management and teamwork strategies that truly reflect the benefit of synergy and optimise the limited resource.

3.4.3 Speed of Motor

As general practice, the speed of the motor can be controlled using pulse-width modulation (PWM) with value ranged from 0 to 255. The motor speed of Arduino Robot may be different even though the values of the both motors (the left and the right) are set to the same value. This issue can be rectified by adjusting the tuneable screw on the Motor Board.

Nonetheless, I found out that the reverse speed of Right Motor, sometimes, may be dissimilar to the speed of left Motor. Therefore, in order to command the robot to turn Right, we need to set the speed of the right motor faster than the

speed of the left motor, e.g. Speed of the Right = -180 (negative means reverse) and Speed of the Left = 90 for turning the Robot to the Right. Surprisingly, this issue suddenly disappears after using the Arduino Robot for few weeks. One possible explanation is that the hardware might be unstable and need some time to “warm it up”.

Besides, it is found that the speed with pulse-width modulation (PWM) lower than 50 is insufficient to make the robot move smoothly. Thus, a commended average speed to control the robot is from 120 to 200.

Another practical issue of Arduino Robot is that the robot may incapable of moving straight using same PWM value. Again, I suspect that this is due to the limitation of the hardware. One possible solution to address this potential issue is to accelerate the speed of the robot instead of applying constant speed. The speed of the robot can be accelerated by increasing the speed of robot gradually. An example program to accelerate the speed of robot is that stated as follows:

ArduinoRobotMotorBoard.cpp

```
speed = maxspeed;
switch(position){
    case 0b10000:
          balance = balance4;
              //  speed = 150;
    break;
    case 0b11000:
        dangerous = 0;
        balance = balance3;
      //  speed = 150;
    break;
    case 0b01000:
        dangerous = 0;
         balance = balance2;
      //   speed = 150;
    break;
    case 0b01100:
        dangerous = 0;
         balance = balance1;
      //   speed = 150;
    break;
}
if (float_speed != speed || float_balance != balance){
    if(float_speed < 50)
        float_speed = 51;
    else if(float_speed < speed && ++update > (maxspeed - (speed - float_speed))*ceil(float_speed/accelerate)){
        update = 0;
        float_speed += ceil((speed - float_speed)/5);
    }
    else if(float_speed > speed)
        float_speed = speed;

    if(float_balance < balance)
      float_balance++;
    else if(float_balance > balance)
        float_balance = balance;

    motorsWrite(float_speed  + float_balance, float_speed );
}
```

Figure 3.4 Example program to control the speed of Arduino Robot

Firstly, the desired maximum speed of the robot to be reached is stored using a variable named **speed**. After that, a **switch** function is used to assign the desired value (**balance**) based on the signals of IR sensors (i.e. **position**). This value (**balance**) will be used to adjust the speeds of motors in such ways that the robot will follow the given line. Two adjustable variables of **float_speed** and **float_balance** will be increased or decreased if either of them is not equal to the desired **speed** or **balance** value (line 385). **float_speed** and **float_balance** are the current speed and adjustable speed of motors, respectively.

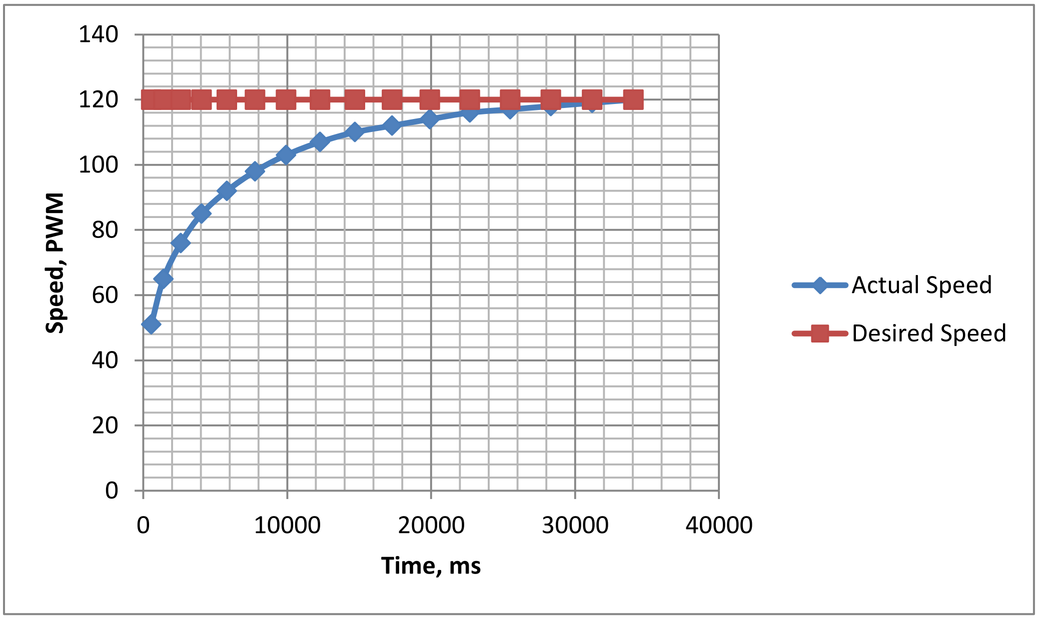


Figure 3.5 The simulation of the controlled PWM

Since the PWM value lower than 50 is insufficient to make the robot moves, I intentionally set the current speed (i.e. **float_speed**) to 51 if the value is less than 50. After that, the current speed will be updated once when the value of **update** is more than **(maxspeed – (speed – float_speed))*ceil(float_speed/accelerate).** The variable of **accelerate** is a tuneable parameter that used to control the duty cycle of the update. To tune the parameter of **accelerate**, we can start with **accelerate = 1**. After that, we can increase the value gradually until the response of the robot is stable. Since **speed** equals to **maxspeed** in this example, the expression can be simplified as (**float_speed)*ceil(float_speed/accelerate)**.

Current speed will be updated using the equation of **float_speed += ceil((speed – float_speed)/5)**. The value of 5 is intuitively set. Nonetheless, the higher the value,

the amount of the update will be reduce, and vice versa. This equation ensures that the incremental of speed will be gradually decreased so that the movement of robot can be more stable. The simulated PWM is illustrated as Figure 3.5.

3.4.4 Design of Robot

Arduino Robot consists of two wheels and two Ball casters. This design may cause one of the wheels becomes unable to contact the horizontal surface properly if the horizontal level of the two Ball casters is slight higher than that of the wheels. One possible solution for this matter is to remove one of the Ball casters.

3.4.5 Motor Control

In order to control the motor and to process the IR signals via Motor Control, we need to write our own Firmware. A comprehensive guidance regarding this matter can be found at the Arduino official website of http://arduino.cc/en/Reference/RobotMotorWriteYourOwnFirmware. Thus, in the following content, I am going to highlight the important steps to accomplish this task.

First, we need to open the C source file (i.e. **ArduinoRobotMotorBoard.cpp**) and Header file (i.e. **ArduinoRobotMotorBoard.h**) of Robot Motor using administrator's account (e.g. right click, then select "run as administrator") so that we can amend the script to suit our needs. These two files are located at directories of **c:/…/Arduino/libraries/Robot_Motor**.

New Mode

Second, we need to define a New Mode at the header files of Arduino Robot Motor and Arduino Control (i.e. ArduinoRobotMotorBoard.h and ArduinoRobot.h). A standard form to define a New Mode named **MODE_NEW** is stated as follows:

```
//motor board modes
#define MODE_SIMPLE 0
#define MODE_LINE_FOLLOW 1
#define MODE_ADJUST_MOTOR 2
#define MODE_IR_CONTROL 3
#define MODE_NEW 4
```

Note:

The name of new mode is always started with **MODE_** followed by a name. The number, however, must be unique compared to the previous defined mode. One

rule of thumb for assigning a unique number to our new defined mode is using the next number of the previous number.

Third, after saving the amendment of **ArduinoRobotMotorBoard.h**, we need to program the new mode by amending the content of **ArduinoRobotMotorBoard.cpp** to suit our needs as follows.

setMode()

When users call the setMode() function in **Robot Control**, Robot Motor Board will execute the script that written in the **void RobotMotorBoard::setMode(uint8_t mode)**. Thus, we can add relevant program when particular "Mode" is requested by the Robot Control. For instance, when the MODE_LINE_FOLLOW is requested using **Robot.setMode(MODE_ LINE_FOLLOW);** in the Robot Control, the Arduino Robot will calibrate its IR sensors. This is because in the **setMode()** of default **ArduinoRobotMotorBoard** program, we have the following script:

```
if(mode==MODE_LINE_FOLLOW){
        LineFollow::calibIRs();
}
```

In other words, we can extend script in the setMode()so that Motor Control board will response accordingly. For example:

```
if(mode==MODE_NEW){
    // Action when MODE_NEW is received;
}
```

Command

Additionally, we can use **Command** definition to enrich our program. Again, we need to define our new **Command** in both ArduinoRobotMotorBoard.h and ArduinoRobot.h using unique number, as that demonstrated to define new mode.

New Function

After having our new Command, we can design the ways that we want to use for Arduino Robot to communicate with the Motor Board using a new function.

Firstly, we need to let our Arduino Robot (i.e. Control Board) knows that we have this new function by defining the new function in its header file. The

location of the new function should be the same as the function of **void begin();** in the header file.

Secondly, we need to create a new .cpp file in the directory of Control Board. One example is stated as follows:

```
#include "ArduinoRobot.h"
#include "EasyTransfer2.h"

void RobotControl::KSCHIA_LINE(int MaxSpeed)
{
    messageOut.writeByte(COMMAND_KSCHIA);
    messageOut.writeInt(MaxSpeed);
    messageOut.sendData();
}
```

Note: Three important functions are used to send information from Control Board to Motor Board, i.e. **messageOut.writeByte()**, **messageOut.writeInt()**, **and messageOut.sendData()**.

messageOut.writeByte() is used to send the **Command** information that we have defined previously so that Motor Board knows how to manage the coming information.

messageOut.writeInt() is used to send information in terms of integral form to Motor Board. Normally, this information is for adjustable parameters. Nonetheless, this function is optional.

Lastly, all information is ended by the function of **messageOut.sendData()** that sends all the information to the Motor Control Board.

ArduinoRobotMotorBoard.cpp – parseCommand()

When Arduino Control Board sends a Command information via **messageOut.writeByte()**, the function of void **RobotMotorBoard::parseCommand()** in ArduinoRobotMotorBoard.cpp will be executed. A switch() function is used to select the required Command. Thus, we need to add a new case with the name of our desired command.

For example, if we send a command named COMMAND_KSCHIA using the function of messageOut.writeByte(COMMAND_KSCHIA); in our Control Board, we need to add the case of the command name as follows:

```
case COMMAND_KSCHIA:
        // Action when COMMAND_KSCHIA is received;
break;
```

If in our Control Board, we send one more information (e.g. MaxSpeed) using the function of messageOut.writeInt(MaxSpeed);, we need to store that information immediately using the function of messageIn.readInt();. The following script shows the program used to store the information:

```
case COMMAND_KSCHIA:
        int maxspeed = messageIn.readInt();
        // Action when COMMAND_KSCHIA is received;
break;
```

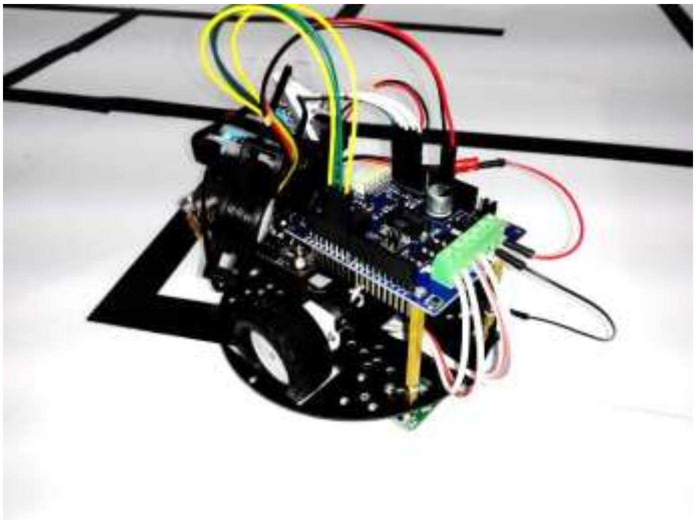
After that, we can add our desired program such as processing the information, read IR sensors, control the speed of motors etc in the next line of the script.

3.5 Summary and Recommendation

Even though Arduino Robot is much expensive than 3pi robot, 3pi robot appears to be more suitable for line following due to its purpose of existence. Arduino Robot, on the other hand, is much complex due to its multi-function and feasibility to interface with other sensors and devices. Thus, for competition or line following demonstration, apparently, 3pi robot is much suitable; while for learning and experiment purpose, Arduino Robot provides more choices.

Chapter 4 Getting Started with FPGA

Eng Pei Chee



4.1 Introduction

This chapter provides a quick guide to get started with FPGA board for controlling a line following robot using **DE0-Nano Development and Education Board** (**Cyclone IV EP4CE22F17C6N**). Relevant topics are organized as follows. First, Altera software installation and USB blaster driver installation are presented. After that, a program using FPGA board is demonstrated. Lastly, a line following example project is provided.

4.2 Altera software installation

This section provides essential procedure for installing Altera Complete Design Suite. Subscription edition, version 14 is used. The latest complete software is available from the Download Center of the Altera website as that listed below:

- Subscription edition: http://dl.altera.com/14.0/?product=#tabs-2
- Web edition: http://dl.altera.com/?edition=web

Alternatively, the Altera Complete Design Suite DVDs are available for request via Altera IP and Software DVD Request Form at:

- https://www.altera.com/literature/quartus2/q2acds-dvd.jsp?GSA_pos=1&WT.oss_r=1&WT.oss=DVD%20request

4.2.1 Downloading Altera Software

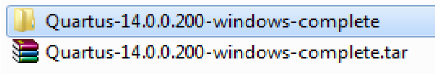
To download the Subscription edition of Altera software, first, go to the abovementioned link. If you do not have license file, please continue the following procedure using **Web edition**.

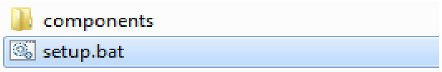
Next, choose **Combined Files** and download the complete software (size: 10.8GB). The download might take time since the complete version file includes all Altera Software and all device family support.

To reduce the download file size, you may choose **Individual Files** and check **ONLY** the **Cyclone IV device support** package. You can still add the other device support packages (e.g. Aria, Cyclone V, MAX II, MAX V, Stratix IV, Stratix V) after the installation is completed. However, the related instruction of the Individual Files installation is not covered in this chapter.

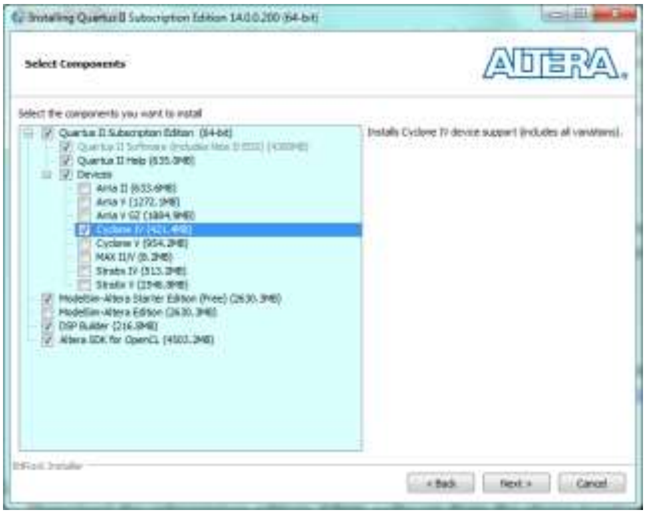
4.2.2 Installing Altera Software

Once the complete Altera software (from Combined Files) has been downloaded on the computer, unzip the .tar file, and then double-click the **setup.bat**. The computer will now prepare for the installation.

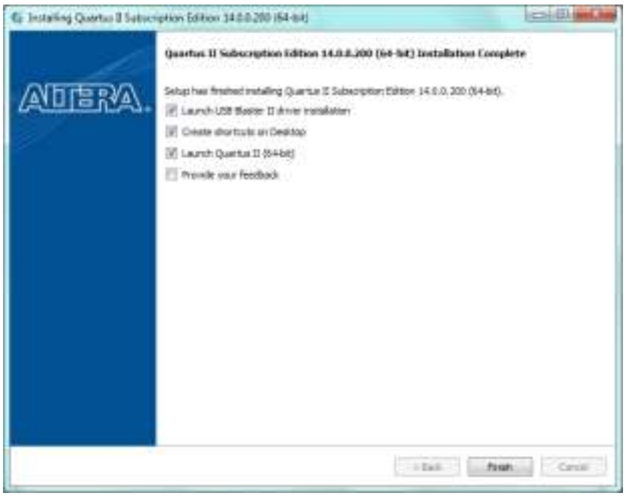




Next, follow the installation instructions. If you prefer a smaller installation size, you may select only **Cyclone IV** device package as follows.



Installation might take some time. Below screen will prompt up when installation is finished.



4.2.3 Installing Altera USB – Blaster Driver

Once the Altera software has been installed, the Altera USB-Blaster or Altera USB-Blaster II “**download cable driver”** must be installed so that FPGA devices can be installed using Quartus II software. The drivers require **manual installation** so that the cable can be properly recognized. Follow the instructions below for the driver installation:

i. Connect a USB cable between the DE0-Nano board and a USB port on the computer that contains Quartus II software.

Skip obtaining driver software from Windows update:

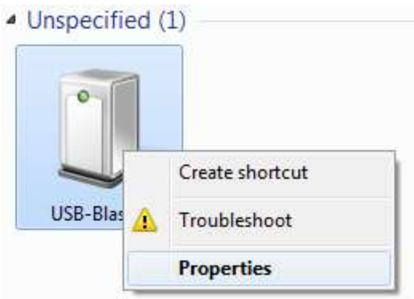
ii. The computer will recognize the new hardware connected to its USB port, but it will be unable to proceed as it does not have the required driver. If the USB-Blaster driver has not yet been installed, the New Hardware Wizard will appear.

iii. If the desired driver is not available on the Windows Update Website, then please click “**Skip obtaining driver software from Windows Update**” to response the question asked. If USB-blaster device driver was not successfully installed, then manually installing the driver is needed.

Manually installing the USB-blaster device driver:

iv. Open “**Devices and Printers**” – choose **Start > Devices and Printers**. You should see the USB-Blaster in the **Unspecified category**.

v. Right-click the USB-Blaster and choose “**Properties**”.



vi. Go to “**Hardware**” tab and click “**Properties**”.

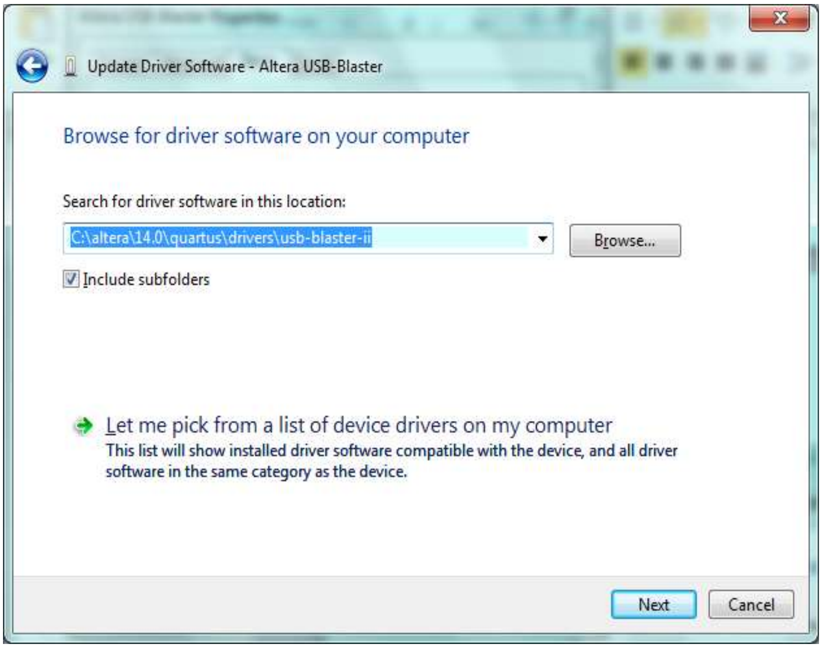
vii. Under the “**General**” tab, click “**Change Settings**”.

viii. In "**General**" tab, click "**Update Driver**".

ix. Select "**Browse my computer for driver software**".

x. Click "**Browse**", and browse to the **<Path to Quartus II installation>\drivers\usb-blaster** directory. Select the "**Include subfolders**" option, and click "**Next**".

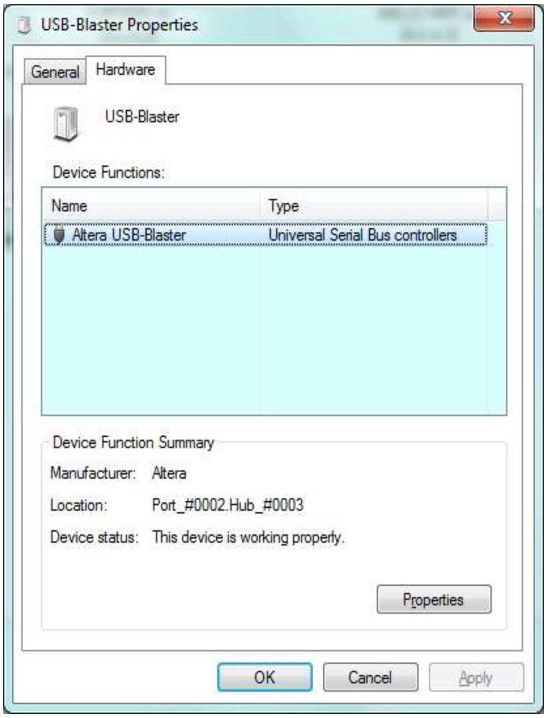
Note: Do not select the subfolder of x32 or x64 directories.



xi. If "**Would you like to install this device software**" is prompted out from the Windows security, please click "**Install**". The installation wizard will guide you through the installation process.

xii. When installation is finished, the "**Windows has successfully updated your driver software**" dialog box appears, and then please click **Close**.

xiii. The device should work properly now.



4.3 DE0-Nano Demonstration Program

This section provides specialized procedures for creating a simple demonstration program to be run on **DE0-Nano development board**. Basic knowledge on FPGA board, basic concepts of programmable logic and basic HDL programming skills are required in this tutorial.

4.3.1 Creating a New Quartus II project

Quartus II project is a set of files that describe the information about your FPGA design. The **Quartus II Settings File (.qsf)** and **Quartus II Project File (.qpf)** are the primary files in the project.

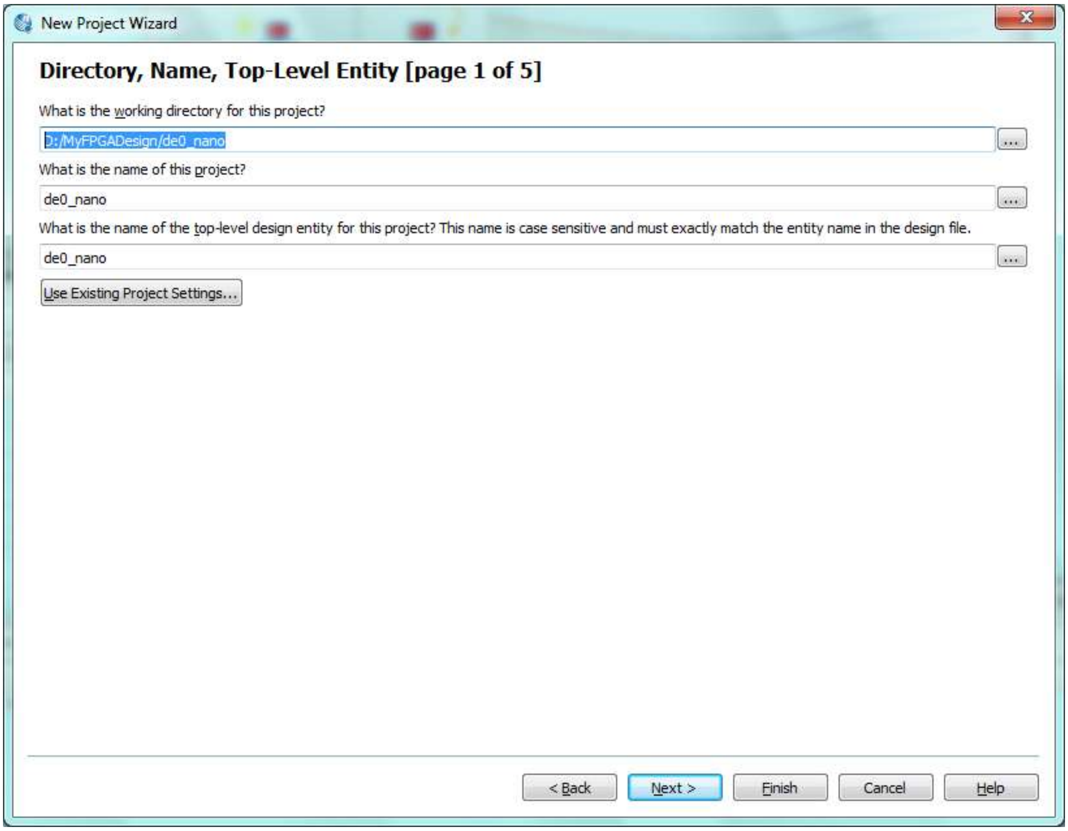
i. Launch Quartus II software. (Choose **Run the Quartus II software** if license has not been bought)

ii. Open the "**New Project Wizard**": select **File > New Project Wizard**.

iii. Click **Next** when the **Introduction** page opens.

iv. In the **"Directory, Name, Top-Level Entity"** page, choose a **working directory** for this project, i.e. ***D:/MyFPGADesign/de0_nano***. After naming the project and top-level entity as ***de0_nano***., please click **Next**.

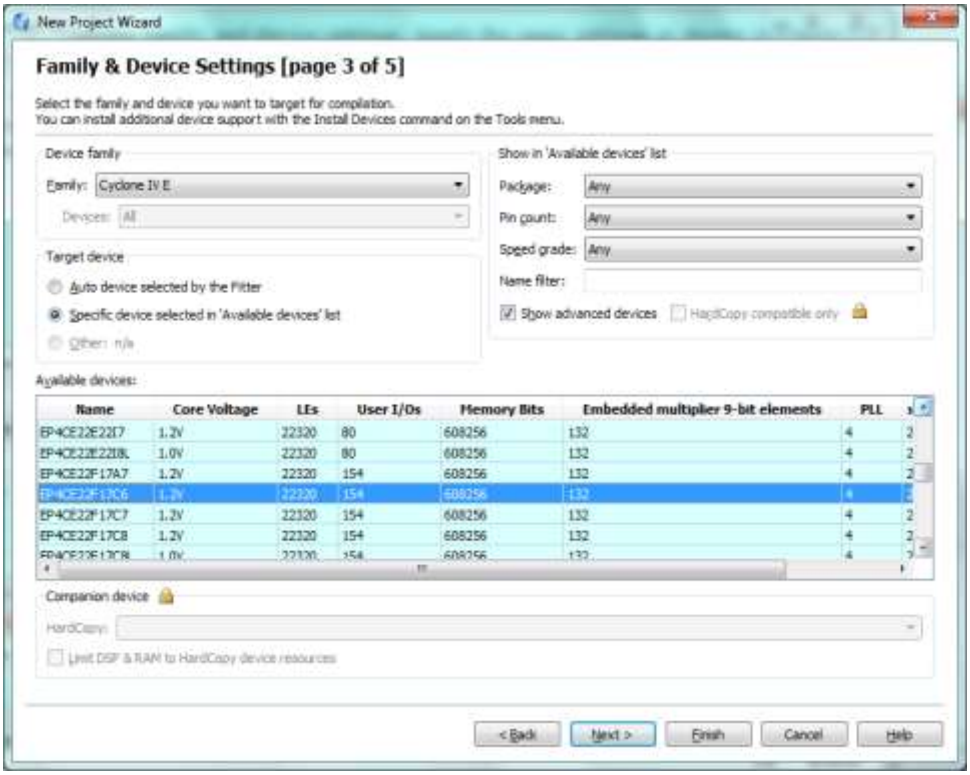
Note: Make sure the path does not contain any spacing.



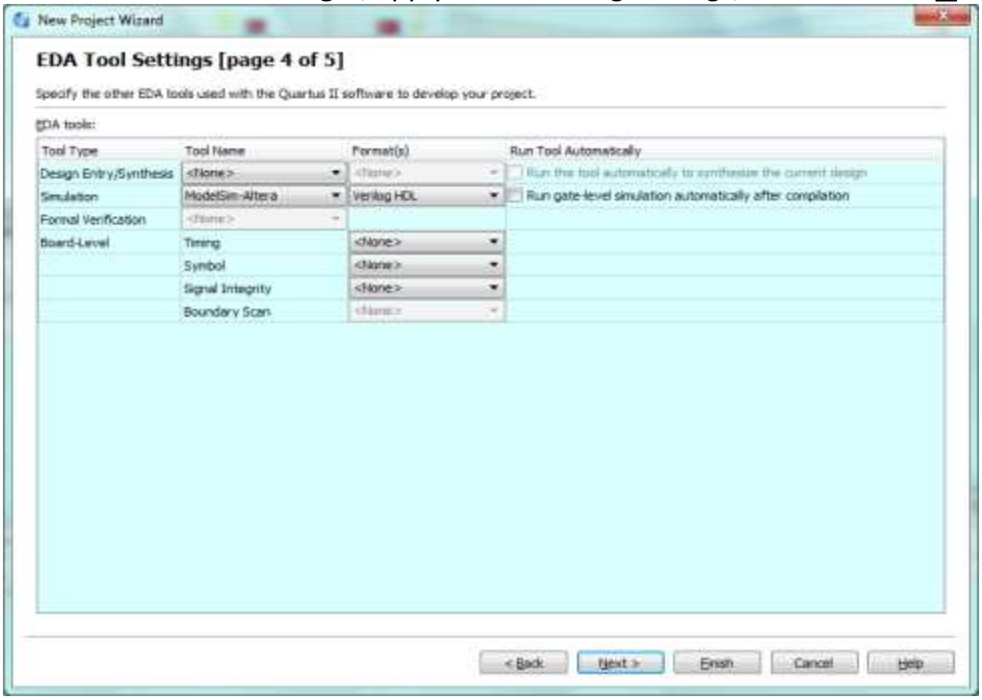
v. Next, **"Add Files"** page will appear for you to select the design files that you may want to include in the project. However, please click **Next** as we are not going to include any design file in the demonstration.

vi. In the **"Family & Device Settings"**, please apply the same settings as that shown in following figure, and then click **Next**.

Note: Please make sure the "**available devices**" is correctly chosen.



vii. In "**EDA Tool Settings**", apply the following settings, and then click **<u>N</u>ext**.



viii. A summary will be displayed. Click **Finish** to end new project wizard.

4.3.2 Importing Pin Assignment

Download the **"DE0_Nano.csv"** pin assignment file from the below link: https://www.dropbox.com/s/f3d6wn4ziohzna7/DE0_Nano.csv

Import the pin assignment to your project:

i. Choose **Assignments > Import Assignments**.
ii. Browse to the **"DE0_Nano.csv"** file which you have previously downloaded.
iii. Click **OK** to import the assignments.

4.3.3 Designing a simple Verilog HDL file

A simple demonstration program written in HDL Verilog will be shown in this section.

i. Create a new "**Verilog HDL File"** in the Quartus II project created in the previous section. Choose **File -> New -> Design Files -> Verilog HDL File**.
ii. **Save** the file as "**de0_nano.v**" i.e. **File -> Save As…**. This will be the top-level module for the project.
iii. Type in the following code, choose Program I or Program II:
    a. **Program I**: use push button **KEY0** to perform left-shift operation which the value is then shown in **LED0-LED7**.

```
module de0_nano (
	LED,	// LED
	KEY	// KEY
);
//=======================================================
//  PORT declarations
//=======================================================
output		[7:0]		LED;	// LED
input		[1:0]		KEY;	// KEY
reg [7:0] temp = 8'b0000_0001;		// INTERNAL WIRES & REGISTERS
//=======================================================
//  Shifting
//=======================================================
always @(negedge KEY[0])
begin
	temp = {temp[6:0], temp[7]};
end

assign LED = temp;
endmodule
```

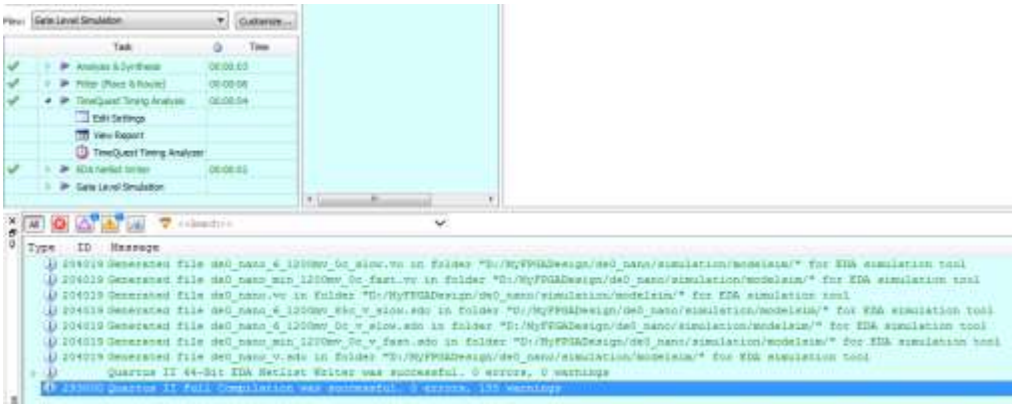
b. **Program II**: perform left-shift operation every 0.5s. Push button **KEY0** is used to reset the board.

```
module de0_nano (
        CLOCK_50,   // CLOCK
        LED,       // LED
         KEY       // KEY
);
//=======================================================
//  PORT declarations
//=======================================================
// CLOCK
input                                   CLOCK_50;
// LED
output                   [7:0]          LED;
// KEY
input                    [1:0]          KEY;
// INTERNAL WIRES & REGISTERS
reg [24:0] counter = 0;
reg [7:0] temp = 8'b0000_0001;
 //=======================================================
//  Shifting
//=======================================================
always @(posedge CLOCK_50)
begin
        if (! KEY[0])
        begin
                counter = 0;
                temp = 8'b0000_0001;
        end

        counter = counter + 1;
        if(counter >= 25_000_000)
        begin
                temp = {temp[6:0], temp[7]};
                counter = 0;
        end
end

assign LED = temp;
endmodule
```

iv. Perform full compilation for each program. Choose **Processing > Start Compilation**. Message “**Quartus II Full Compilation was successful**” will be displayed when the program has been compiled successfully.



4.3.4 Configure DE0-Nano Development Board

You can now configure the DE0-Nano FPGA development board once the project is fully compiled.

i. Connect your DE0-Nano FPGA development board to your computer via a USB cable. The USB cable is used as the USB-blaster programming cable and the power source of the FPGA board. Optionally, you can also power up the board via an external power supply.

ii. Launch the **Programmer**:
Choose **Tools > Programmer**.

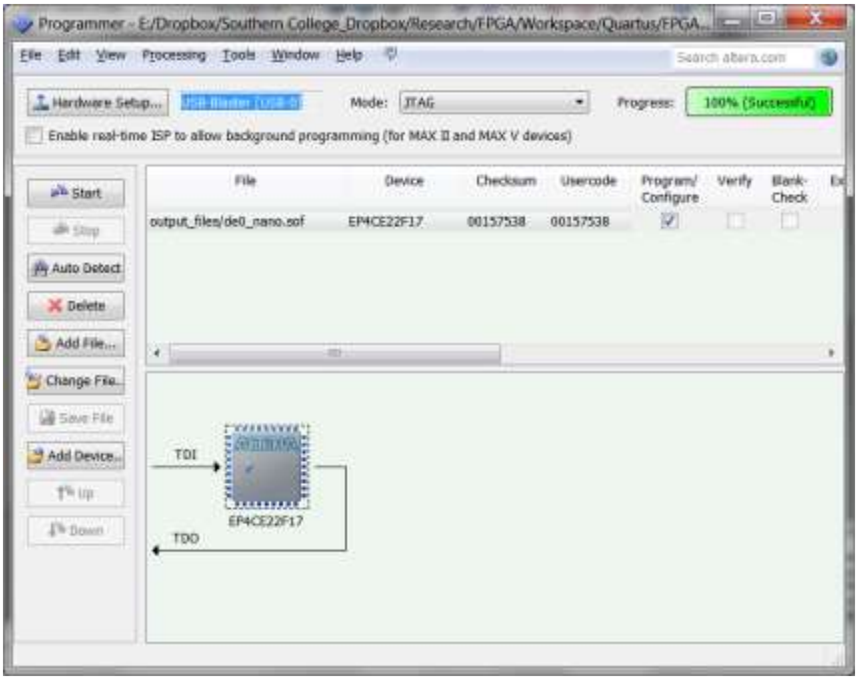
iii. **Set up the programming hardware**:
Click on the “**Hardware Setup**” on the upper left. Next, in the “**Hardware Settings**” tab, choose **USB-Blaster** as “**Currently selected hardware**”, and then close the “**Hardware Setup**”.

iv. **Add the .sof programming file**:
Click “**Add File…**” on the left hand side, and then browse for the .sof programming file.

Note:

- The .sof programming file should be located in “**output_files**” folder.
- If you are using Web Edition Quartus II, a temporatry .sof programming file will be generated. In this case, the “**Programmer**” Window must be opened in order to keep the system working on the FPGA board.

v. **Configure the FPGA board**:
Check the "**Program/Configure**" checkbox, and then click **Start** to upload the .sof file to the FPGA board. The "**Progress**" bar will show 100% once the system is successfully downloaded.

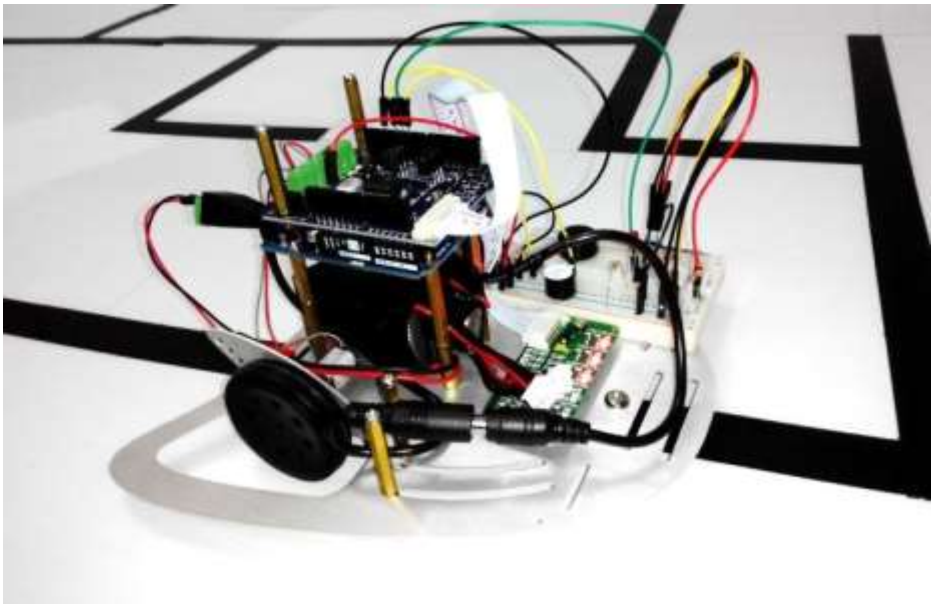


4.3.5 Test the FPGA Board

Now, you can test the demonstration code that stated in Section 4.3.3 using FPGA Board. If the Program 1 is uploaded to the FPGA board, a LED shift operation in **LED0-LED7** on the FPGA board can be observed when button **KEY0** is pressed.

Chapter 5 Line Follower Robot with Arduino UNO

Chia Kim Seng



5.1 Introduction

Deploying a line follower robot from pieces of components not only improves learners' understanding about the robot, particularly from electronic and mechanical hardware perspectives, but also increases the satisfactory level of the learners once the robot is completed. In fact, the knowledge and experience are transferable for learners to be implemented in different projects that involve microcontroller, motor control, sensor, etc.

Thus, this chapter provides the procedure needed to self-deploy a line follower robot. The scope includes materials purchases, background, and programming.

5.2 Materials

A basic line follower robot consists of five indispensable parts, i.e. frame, motor module, sensor module, control module, and power supply.

5.2.1 Frame

The frame of robot determines the structure of a robot. The frame should be big and strong enough to store and protect the critical components (e.g. motor, sensor, and controller) of a robot. Besides, the frame also should be feasible in such ways it can be customized to suit user's needs.

There are abundant ready-to-assemble frame in the market. Generally, these robot frames will be sold together with two motors. The main reason of this practice could be due to the intention of sellers to help users to avoid the difficulty to install motors to that particular frame.

Nonetheless, users should pay attention to the maximum speed of given motors. This is because motors with slow speed may have excellent precision but they are unsuitable for racing purpose. High speed motors that may be good in racing, on the other hand, may be less robust. Consequently, it is very difficult to control such less robust motors.

In this project, an aluminium robot chassis complete set that contains two geared DC motors was used as the frame of the robot.

5.2.2 Motor Module

One main component of a Motor module is H-bridge. H-bridge (http://en.wikipedia.org/wiki/H_bridge) is used to change the direction of current flow through a motor in such ways that the rotation direction of a motor can be controlled.

Besides, a current regulator is needed to regulate the amount of current needed for a motor to maintain a required speed.

Thus, a motor module is a combination of H-bridge (i.e. to control direction) and a current regulator (i.e. to control the speed via controlling the amount of current).

The Motor module (or Motor driver) that used in this project was 3Amp Motor Driver Shield (SHIELD-2AMOTOR).

5.2.2.1 Motor Module - Interface

To interface with a motor module, generally, we need at least two pins per motor. One of the pin will determine the direction of a motor, i.e. forward or reverse. This can be done using a simple digital signal of HIGH for forward or LOW for reverse, or vice versa.

Next, Pulse-Width-Modulation (PWM) is used to control the current regulator in ways that the speed of motor can be controlled as desired. To generate high performance PWM, a controller with PWM or Timer function is needed so that the PWM can be generated independently from its principal program.

5.2.3 Sensor Module

Sensor module, generally, consists of three or more set of infrared (IR) emitters and receivers. Each IR receiver has its corresponding IR emitter. IR emitter emits IR energy, while IR receiver senses the amount of IR energy. The amount of IR energy that received by the IR receiver will be used to determine the position of a robot. The term of IR sensor is used to represent a combination of one IR emitter and one IR receiver.

The signal of IR receiver is in analog. Generally, the signal ranges from 0V to 5V. In the following example, the signals of 3 IR sensors were digitalised to 0 and 1 with cut-off value of 3.3V. The signal of ON/HIGH or 1 indicates that the IR sensor is on the line, while 0 indicates that IR sensor is out of the line. Table 5.1 summarise the actions needed upon the status of the three IR sensors.

The sensors module that used in this project was Auto-Calibrating Line Sensor (LSS05).

(left) IR signal (right)	Sensor(s) on the line	Action
000	All are out of line	Stop robot
100	Only the Left IR sensor	Turn to the Left
110	Both Left and Mid IR sensors	Adjust to Left slowly
010	Only Mid IR sensors	Forward
011	Both Right and Mid IR sensors	Adjust to Right slowly
001	Only the Right IR sensor	Turn to the Right

Table 5.1 Actions needed upon the status of the IR sensors

5.2.4 Control Module

A controller is considered as the "brain" of a line follower robot. This is because a controller is needed to analyse the signals from sensor module in order to determine the present position of the robot, and then instruct the motor module accordingly so that the robot can perform properly e.g. follow a line properly.

In this chapter, Arduino UNO is proposed to be implemented as a control module due to its ready-to-use feature. The communication between a computer and Arduino UNO is similar to that used for Arduino Robot (Chapter 3), i.e. Arduino IDE. Thus, the following content focuses on some important areas of the programming.

5.2.5 Power Supply

Since Arduino UNO consists of internal regulator and a 5V A-B USB pin, users can use either a 7.4V li-ion/li-po battery or a 5V power bank to power the controller. In this project, 7.4V li-po battery (900mAH) is used to power the Arduino UNO using DC Jack (Male) to DG126 Converter between the battery and the controller.

5.3 Programming

Initially, we need to include the header file of **Arduino.h** so that we can "call" those user-friendly functions from Arduino library, i.e. pinMode(), digitalRead(), analogWrite(), and digitalWrite().

5.3.1 Define Input or Output (I/O)

After connecting the motor module and the sensor module to the controller of Arduino UNO (http://arduino.cc/en/Main/arduinoBoardUno), we need to define the pin or connection as either Input or Output.

For motor module, apparently, Arduino UNO only sends signals to the module. Therefore, those pins connected to the motor module should be defined as Outputs. This can be done using the function of **pinMode(PIN, OUTPUT);** in the void setup(){ }.

For IR sensor module, on the other hand, Arduino UNO basically only read the signals from the module. Thus, we need to define the pins that connected to the module as Input. This can be done using the function of **pinMode(PIN, INPUT);** in the void setup(){ }.

5.3.2 digitalRead( ) Vs analogRead( )

In Arduino UNO, the function of digitalRead(**PIN**) will return a digital signal of HIGH or LOW. These HIGH or LOW signal is useful in helping users to determine the position of IR sensors.

The function of analogRead(**PIN**), on the other hand, will return an analog value from 0 to 1024. To read analog value, users need to connect their analog device to the analog pin of Arduino UNO, i.e. A0 to A5.

5.3.3 digitalWrite( ) Vs analogWrite( )

In order to generate digital signal of HIGH or LOW, digitalWrite(**PIN**, HIGH/LOW); should be used.

To generate 8-bit PWM, on the other hand, we need to use analogWrite(**PIN**, value); (http://arduino.cc/en/Reference/AnalogWrite) with value ranges from 0 (always off) to 255 (always on). The only pins that can be used for PWM purpose are pins of 3, 5, 6, 9, 10, and 11.

5.3.4 Example Code

This section explains a simple code that used for Arduino UNO to control its motors based on the signals of IR sensors.

5.3.4.1 Initialization

Firstly, the header file named "Arduino.h" is included as follows:

```
#include "Arduino.h"
```

Then, pin 4 and pin 5 are defined as DIR1 and EN1 as the direction of motor 1 and the PWM of motor 1, respectively; and pin 7 and pin 6 are defined as DIR2 and EN2 as the direction of motor 2 and the PWM of motor 2, respectively, as follows.

```
#define DIR1 4
#define EN1  5
#define EN2  6
#define DIR2 7
```

After that, all related I/O are defined accordingly in the setup(), i.e. all pins to motor module are OUTPUT, while all pins from IR sensors (A1 to A5) are INPUT. Besides, the motor condition is initialized as stop mode so that the robot would not run randomly when it is power on. This can be done using digitalWrite() as follows:

```
void setup(){
        pinMode(DIR1, OUTPUT);
        pinMode(EN1, OUTPUT);
        pinMode(EN2, OUTPUT);
        pinMode(DIR2, OUTPUT);

        pinMode(A1, INPUT);
        pinMode(A2, INPUT);
        pinMode(A3, INPUT);

        digitalWrite(DIR1, LOW);
        digitalWrite(EN1, LOW);
        digitalWrite(EN2, LOW);
        digitalWrite(DIR2, LOW);
        digitalWrite(CAL, HIGH);
}
```

5.3.4.2 Read IR sensors

Simple routine actions are programmed in the loop() function so that the robot can read the latest IR signals. According to Table 5.1, there are six possible IR signals' conditions that a robot needs to react with. To determine condition of the IR signals, **if()** and **digitalRead()** functions are used. For instance, the code below tests whether only the Left sensor is on line or not.

```
if(digitalRead(A1)==HIGH && digitalRead(A2)==LOW && digitalRead(A3)==LOW)
```

If the condition is fulfilled, then we shall adjust the speed of motors accordingly so that the robot will move along the line, i.e. only the middle IR sensor is on the line.

Additionally, the output of a particular IR sensor may not in digital form. When this happens, users need to use **analogRead()** function to analysis the analog signal of the IR sensor.

5.3.4.3 Control the speed of motors

The speed of motors can be adjusted using PWM that generated from Arduino UNO. Firstly, we need to connect the PWM pins to the speed control pins of motor driver. After that, we need to use **analogWrite()** function to control the value of PWM. The value of **analogWrite()** ranges from 0 to 255. In other words, PWM of zero implies that the motor should be stopped, and PWM of 255 means the motor is going to rotate with maximum speed.

To turn right, for instance, we can stop the motor on the right hand side, while rotating the motor on the left hand side. However, this kind of turning method does not guarantee the smoothness of the turn. In fact, the speed difference between the motors (right and left) will determine the smoothness of a turn. Thus, one method to optimise the smoothness of a turn is to figure out the best speed difference.

5.4 Summary

Previous content has highlighted that users should ensure that the rotation speed of motors are suitable for their project (Section 4.2). Besides, the details of hardware that involved in building a line follower robot is given for consideration.

Chapter 6 Line Follower Robot with DE0-Nano FPGA

Eng Pei Chee

6.1 Introduction

The following section describes the specialized procedures to self-deploy a line follower robot controlled by a reconfigurable FPGA board. The topic includes hardware construction, line following principles, and Verilog HDL implementation.

6.2 Hardware design

The self-deployed FPGA line follower robot consists of 5 basic components, i.e. Chassis, power supply, sensor module, motor driver and DE0-Nano FPGA board as that shown in Figure 6.1.

6.2.1 Chassis

The chassis is a ready-to-assemble 2-wheel balancing chassis from *DFRobot*. Although it is featured with a powerful metal gear motors that provide a speedy robot, it is less robust and need a great effort to control the motor.

6.2.2 Power Supply

Two power sources are needed in order to power up the line follower robot. One power source is used for the FPGA board that specifies an input voltage ranged from 3.6-5.7 volt. This can be done using four NiMH AAA rechargeable batteries that connected in series. The other power source is is a 7.4V 900mAH Lipo Battery that is used to power the motor driver.

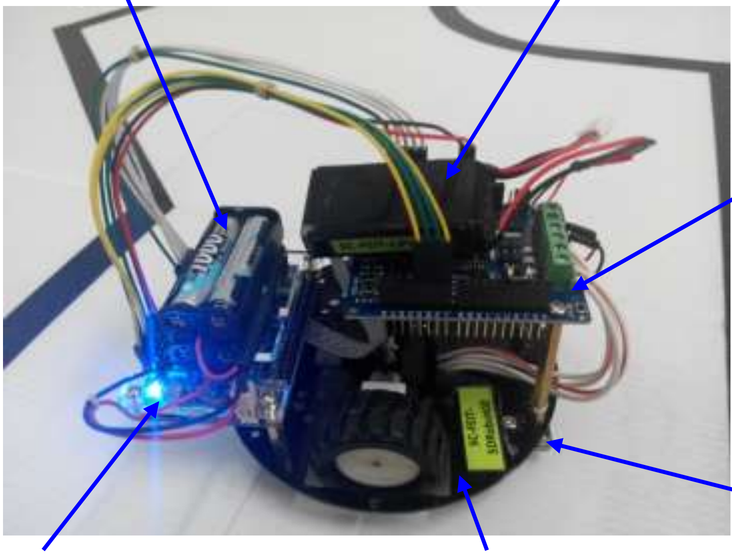


Figure 6.1 Self-deployed FPGA line follower robot

6.2.3 Sensor Module

LSS05 IR line sensor bar that mounted below the chassis is used to differentiate the dark and light surface. LSS05 provides 5 sets of IR transmitter-receiver with digitized output (1 for black, 0 for white) indicating the existence of the line. The threshold to the lightness of line existence will be set when the sensor is calibrated.

6.2.4 Motor Driver Module

A motor driver is needed to drive a DC motor. 2A Motor Driver Shield (SHIELD-2AMOTOR2) is used in this project. This motor driver allows a direct drive of two 7-12VDC bi-directional DC motors with maximum current of 2A.

6.2.5 Control Module

A control module or the “brain” of a robot is needed to receive and analyse the signals from the sensor module, and then instruct the motor module accordingly. The control module is used to receive and analyse the 5 IR digital output signals, and to control direction of the robot via changing the speed of both motors. The

speed of motor is controlled using Pulse Width Modulation (PWM) written in Verilog HDL.

We have explored an alternative way to implement the control module by using DE0-Nano FPGA board. The compact-sized FPGA development board targeting Cyclone IV device provides up to 22, 320 logic elements (Les). It includes two external General-purpose input/output (GPIO) headers to extend designs beyond the DE0-Nano board, on-board memory devices including Synchronous dynamic random access memory (SDRAM) and Electrically Erasable Programmable Read-Only Memory (EEPROM) for larger data storage and frame buffering, as well as general user peripheral with LEDs, toggle switch, and push-buttons.

6.2.6 Interfacing - Putting it all together

Figure 6.2 depicts the hardware connection of all modules mentioned above.

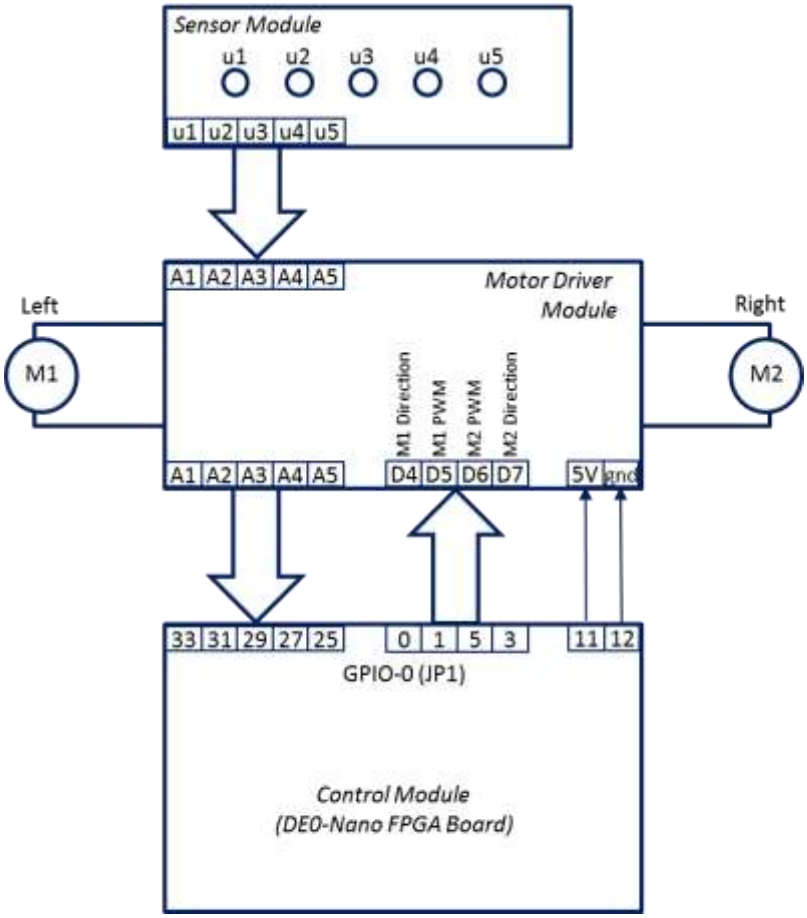
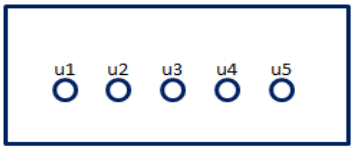


Figure 6.2 The block diagram of the hardware connection

6.3 Principles of Line Following

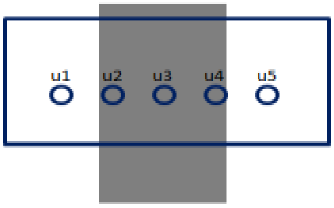
This section describes the principles of how a robot follows a line. There are six general situations that could happen when a robot is following a line as follows:

i. Out of line



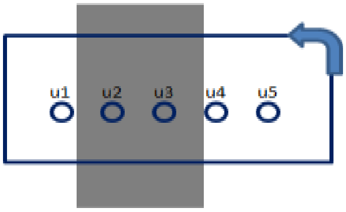
Situation	Sensor Reading	Action
out of line	00000	stop

ii. Being on the line



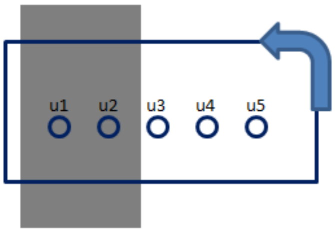
Situation	Sensor Reading	Action
center of line	01110	go straight

iii. Drift right



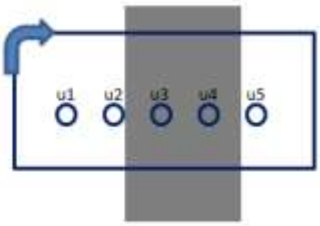
Situation	Sensor Reading	Action
drift right	01100	turn left (slow)

iv. Big drift right



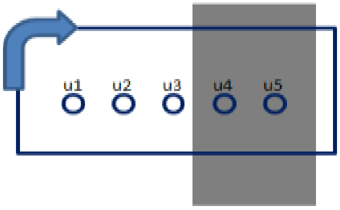
Situation	Sensor Reading	Action
big drift right	11000	turn left (fast)

v. Drift left



Situation	Sensor Reading	Action
drift left	00110	turn right (slow)

vi. Big drift left



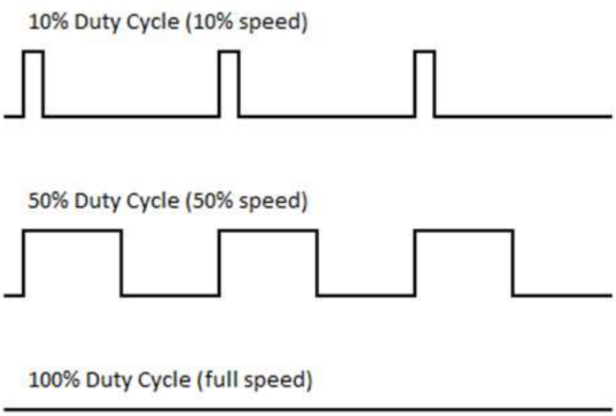
Situation	Sensor Reading	Action
big drift left	00011	turn right (fast)

6.4 Verilog HDL Implementation

Two hardware modules were developed written in Verilog HDL namely **pwm module** for changing the motor speed and **Moore finite state machine line_following module** for controlling the speeds of motor 1 and motor 2 in accordance with the signals of IR sensors.

6.4.1 PWM module

A PWM signal is generated with different duty cycles to change the speed of a motor. Smaller duty cycle means slower speed and vice versa.



The complete code for the **pwm module** that given below is specified for 10 MHz input clock, 8 step count bits and 1ms refresh time or equivalent to 1 kHz PWM output.

The duty cycle is determined by checking the counter value. While it is less or equal to the multiplication of step count and step delay, the PWM output is set to HIGH. Otherwise, the PWM output is set to LOW.

```
module pwm (
	clock,
	stepCountBit,
	pwm,
	pwm_value,
	reset
);
//specification
	//input clcock, clk = 10MHz	//stepCountBit = 8 bits	//refresh time, t = 1ms
	//calculation:
		//wave period, P = t * clk = 1m * 10M = 10 000
		//step count, sc = 2^(#sw) = 2 ^ 8 = 256
		//step delay, sd = P / sc = 10000 / 256 = 39.0625 ~ 39
		//# bit for counter, n = log2(P) = log2(10000) = 13.287 ~14 bits
		//PARAMETER DECLARATION
parameter sd = 39;
//PORT DECLARATION
input			clock;
input [7:0]		stepCountBit;
output			pwm;
input			pwm_value;
input			reset;
//INTERNAL REG & WIRE
reg pwm;
reg [13:0] counter = 0;
//PROCESS
always @ (posedge clock)
begin
	if (reset==0)			counter = 0;
	//calculating duty cycle
	counter = counter + 1;
	if(counter <= stepCountBit * sd)	pwm = pwm_value;
	else				pwm = 0;
	//refresh every 1ms
	if(counter >= 10000)		counter = 0;
end

endmodule
```

6.4.2 Line Following Module

A **Moore Finite State Machine** that illustrated in Figure 6.3 was designed to perform a series of actions, i.e. left-turn, right-turn, forward and stop. This series of actions is used to control a robot so that the robot can follow a given line. The inputs of the circuit are the signals of IR sensors (u1-u5). Two outputs of **m1_pwm** and **m2_pwm** are generated to control the speeds of motor 1 and motor 2.

User may add additional states to solve a more complicated task, i.e. right angle, acute angle or obtuse angle junction turn.

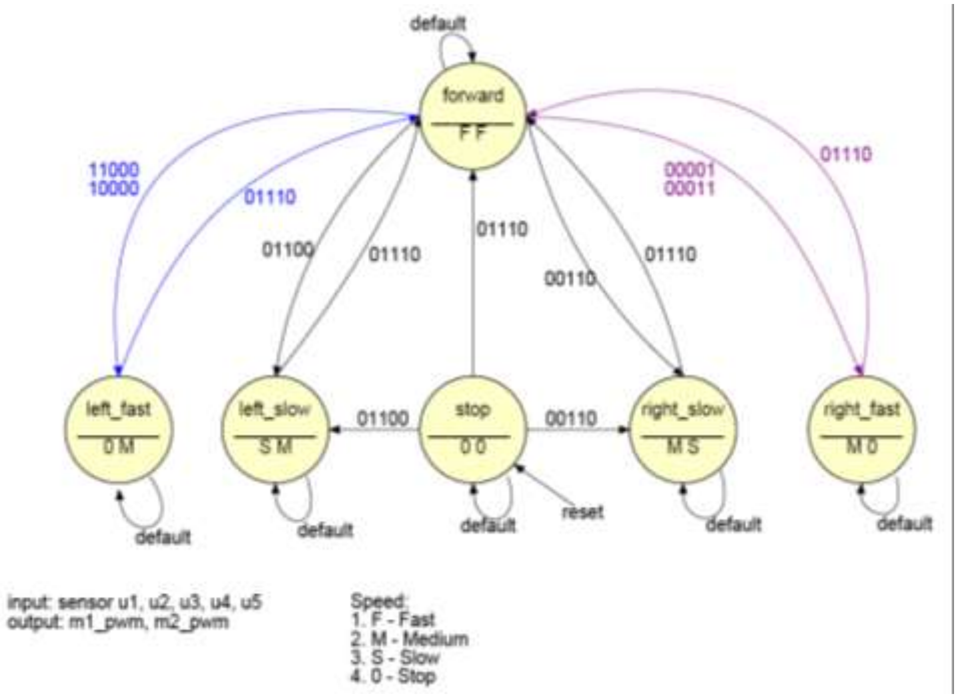


Figure 6.3 Moore Finite State Machine

The complete Verilog HDL code that is given in Appendix B is capable of following a straight line, a curve, right angle and obtuse angle turn.

6.4.3 Top-level module

Before connecting the two modules mentioned above, a 10 MHz frequency is needed to be generated from the system 50MHz clock frequency as the input frequency for ***pwm* module**. One way of doing this is by using the Altera Phase-Locked Loop (ALTPLL) IP core. The following procedure creates a ***pll* module** with ALTPLL IP core.

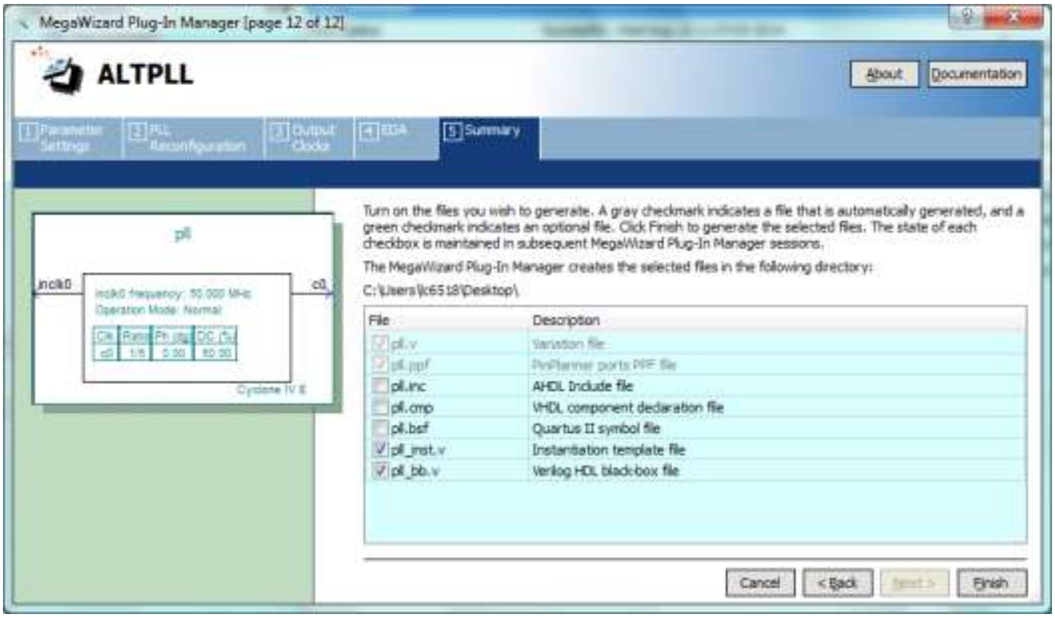
i. From the "**IP Catalog**" window on the right hand side, select **Library -> Basic Functions -> Clocks; PLL and Resets -> PLL**, then double-click **ALTPLL**.
ii. For "**IP variation file name**", name it as "**pll**" and select "**Verilog**" for the "**IP variation file type**". Click **OK** to proceed. "**MegaWizard Plug-in Manager**" will be prompted out shortly.
iii. In the "**MegaWizard**", for "**1 Parameter Settings**":
    - Under "**General/Modes**", key in 50 MHz as the **inclk0** input frequency, and then click **Next**.

- Under “**Inputs/Clock**”, uncheck the option “**Create an ‘areset’ input to asynchronously reset the PLL**” and “**Create ‘locked’ output**”.

iv. Next, select the tab of “**3. Output Clocks**”:

- For “**clk c0**”, check the “**Use this clock**” checkbox, then select the “**Enter output clock frequency**” radio button, and key in **10 MHz** as the output frequency.

v. Click **Finish** to end the settings. A summary page that shown in below figure will be displayed. Click **Finish** to exit the ALTPLL IP core. You should now find 5 files (pll.qip, pll.v, pll_bb.v, pll_inst.v and pll.ppf) generated in the project folder.



The code that stated in Appendix C is used to wrap the ***pwm* module**, ***lineFollowing* module** and ***pll* module** in the **Top-Level module**. To compile and upload the program to FPGA board, please refer to Section 4.3.3 and Section 4.3.4.

6.5 Summary

The robot behaves as planned. It is able to follow any black line with a wide of 2.3cm. The most challenging part in developing the self-deployed FPGA robot is on controlling the speed of the motor. Users need to figure out the best combination of speeds so that the robot can follow a line smoothly.

Chapter 7 Robot Racing

Kang Eng Siew

7.1 Introduction

This chapter aims to provide the essential steps to build a good looking line following course. The necessary materials are mostly available in most stationery shops and they should cost under RM 20, depending on the dimension of the corrugated board. In addition, this chapter discusses the optimization of the program to make the robot zoom down the line at the highest speed possible, with a mixture of gradual turns and sharper turns.

7.2 Building the Line-Following Course

7.2.1 Materials

Before we start to sketch the racing course, necessary materials that used to construct a complete line or maze are stated as follows:

- Large sheet of white corrugated board (plastic) or whiteboard to create the base of the demonstration field (Note: This material is normally available at local office supply stores or stationery supply stores).
- Light-coloured masking tape or some sorts of clips to combine multiple boards if you are using more than one corrugated board to build a demonstration field.
- Black-coloured electrical tapes to create circle lines. Electrical tape works well because it is dark and flexible. Besides, it is easy to be removed.
- Compass or circular objects to sketch circular lines on the demonstration field.
- Drawing tools such as a pencil, eraser and ruler.

7.2.2 Three simple steps

1. Design the racing course on a A4 paper

- The first in building the line-following course is to sketch out your racing design on a piece of A4 paper with precise dimensions and angles by considering the exact dimensions of the corrugated board in the sketching as well as taken into account the circular object that you will use to trace arcs.
- It is recommended to consider the dimension of the robot to provide appropriate borders on the long and short sides of the corrugated board.
- Also, take note on the distance between the electronic systems sensors for the Arduino in order to select the nicely matched electrical tape that will be used to lay the straight lines and circles.
- For 3pi Arduino robot, it is suggested to use ¾ inch electrical tape.

2. Transfer the course onto the corrugated board

- Once the sketching is completed, we need to transfer the initial sketch onto the corrugated board by using a pencil, ruler, compass or circular object.
- Remember to mark the 4” grid on each edge of the board. Nonetheless, the size of grid is dependent on the dimension of your robot. After that, we can start to draw the design within the border.
- It is encouraged to start the drawing with the circular segment with particular arcs, followed by the straight lines.

3. Lay the black-coloured electrical tape on the corrugated board.

- Always keep the electrical tape stretched out tightly in a direction tangent to the path. This will produce a consistent and clean line.
- Each line segment should be made with a cleanly-cut piece of electrical tape. Make each piece as long as possible, overlapping them at the corners and intersections.
- In addition, it is necessary to pay careful attention to the corners so that they come out as clean as possible. This is because any extra piece of tape that sticks out at incorrect angles could cause a robot to confuse.

Figure 7.1 Example of racing design: curves, 90 degree, triangle and straight lines.

7.3 How a Robot Follows Its Path

A robot seeks a guideline to follow a given path. There are three fundamental situations that could happen. The simplest case is: when both sensors are above a guideline, then the robot should go straight. For example: if (right sensor is HIGH, left sensor is HIGH), then go straight / power both motors in the forward direction (Figure 7.2).

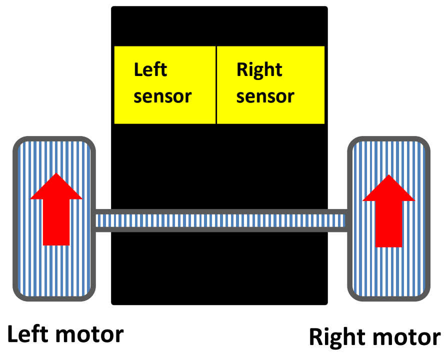


Figure 7.2 Being on the straight line when both sensors detect the black grid.

When approaching a curve, the right sensor loses contact with the line. In order to contact with the line using right sensor again, the robot can slow down or stop its left motor. By doing so, the robot will turn to the left. For example: if (right sensor is LOW, left sensor is HIGH), then move towards left / power only right motor

(Figure 7.3). On the other hand, when the left sensor loses contact with the line, the robot is needed to be turn to the right, i.e. if (right sensor is HIGH, left sensor is LOW) move towards right/ power only left motor.

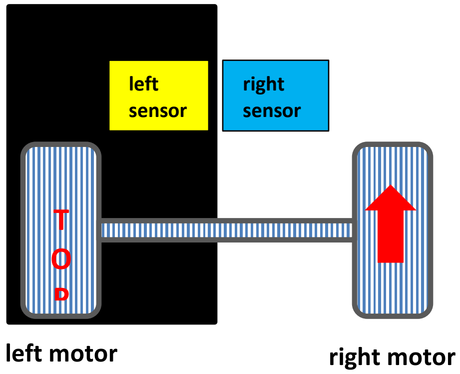


Figure 7.3 Being on the curve where left sensor detects black line and right sensor losses contact with the line - the robot is controlled to turn to the left direction.

If the curve is too sharp where the robot's turning radius is greater than the curve radius of the line, a robot can lose the guideline and even go outside the line. When this happens, the direction of the motors is needed to be changed so that the robot can turn towards to the line (Figure 7.4).

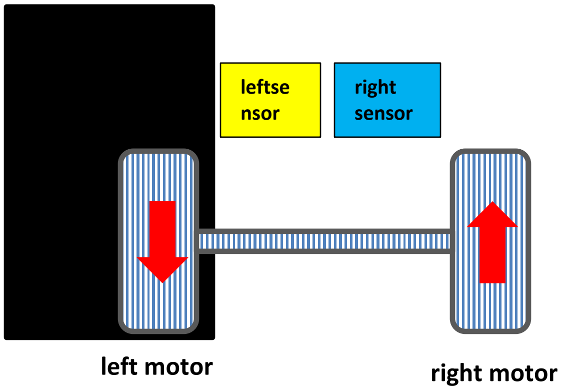


Figure 7.4 Outside of the line where both of the sensors miss the black line.

7.4 Challenges

When designing a robot racing course or demonstration, we should ensure that a reasonable contrast between the line and the corrugated board is chosen and the turns are not cruelly sharp so that the robot is able to follow the given path.

The competition courses usually come in three different general difficulties levels:

1. Easy: The path has gentle and smooth, usually 6-inch radius curves
2. Medium: The lines either cross each other and/or there are up to 90 degrees sharps turn.
3. Hard: The path contains crossings and is often greater than 90 degrees turns.

Normally the contests put a premium on speed in which the robot that finishes the path in the shortest time will be the winner.

7.5 Assignments for Beginners

Student groups are challenged to program robots with IR sensors to follow a black line on a white background. Learning the logic and skills during programming robots helps students improve their understanding of how robots "think", and widens their appreciation for the complexity involved in programming. They test their ideas for approaches to solve the problem and ultimately learn a working programming solution. With these assignments, students are able to think of real-world applications for line-follower robots that use the signal of IR sensors as inputs.

Assignment 1

The first task is the simplest line follower with only straight lines. Your robot is required to follow the strip of black electrical tape on the white background.

First, your robot is located at one end of the continuous black grid. The goal of this assignment is to program or “teach” the robot so that it is capable of moving to the other end. Learners are required to use their C programming knowledge (Chapter 9) to program and control the robot to react accordingly (e.g. turn left, turn right, move straight) so that the robot can moving along the black line.

After that, sharp turns can be included to increase the challenge of the assignment (Figure 7.5).

Tips: You may need to write a test program to understand how the sensors and motors work.

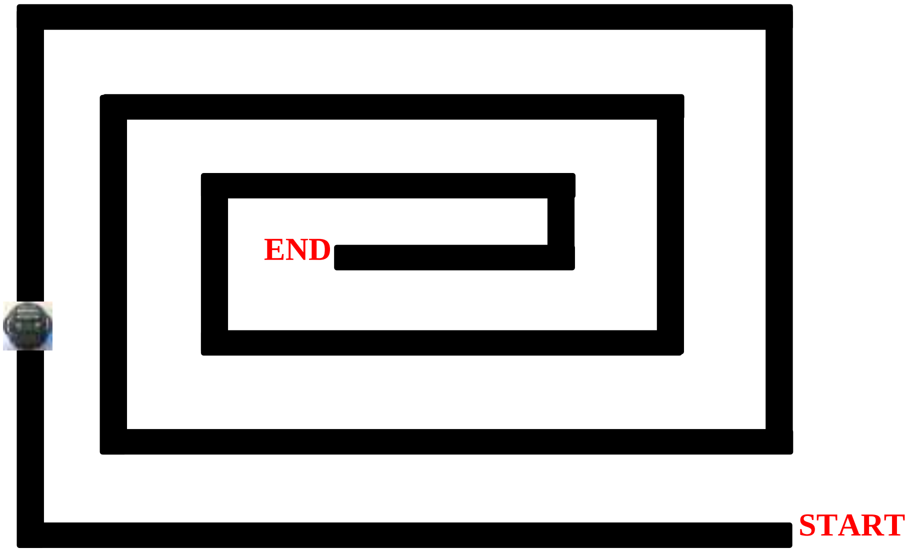


Figure 7.5 Example of simple combination of straight lines and sharp turns.

Assignment 2

Design an assignment for a Line Follower robot learning with a mixture of gradual turns and sharp turns may increase the challenge of an assignment. One challenge is to “train” a robot to move forward while following the curve of the line. The tighter the curves or turns, the slower the robot’s movement is needed to avoid missing the turns. The objective would be to maintain a gray level indicating that the sensor is at the edge of the line, following either the right or left edge as the robot moves forward.

Figure 7.6 Example of simple combination of straight lines and curves.

Assignment 3:

Now, we can get the maps to be a bit more complex for deeper learning. The entire map is a grid of black lines, with a black space on the grid (Figure 7.7). The task is to "train" the robot so that the robot is able to walk around, find the black space, and then park on the black space.

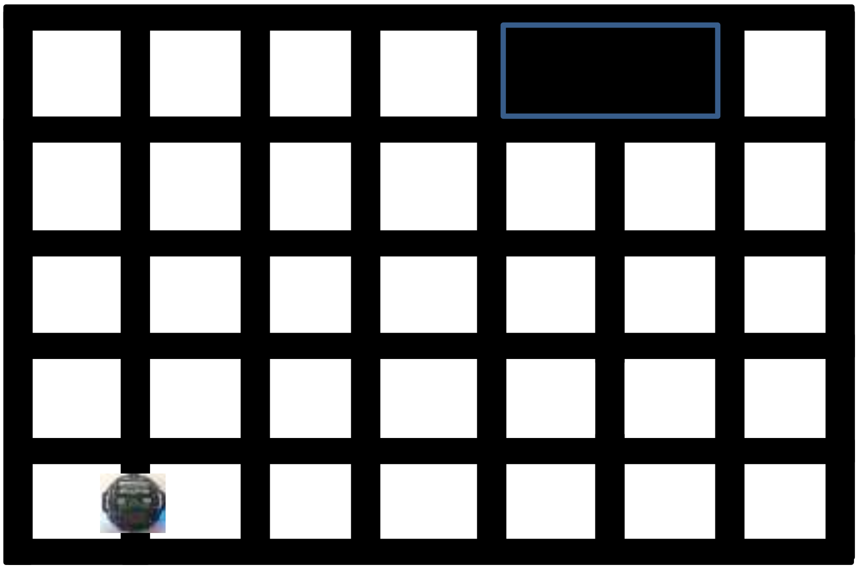


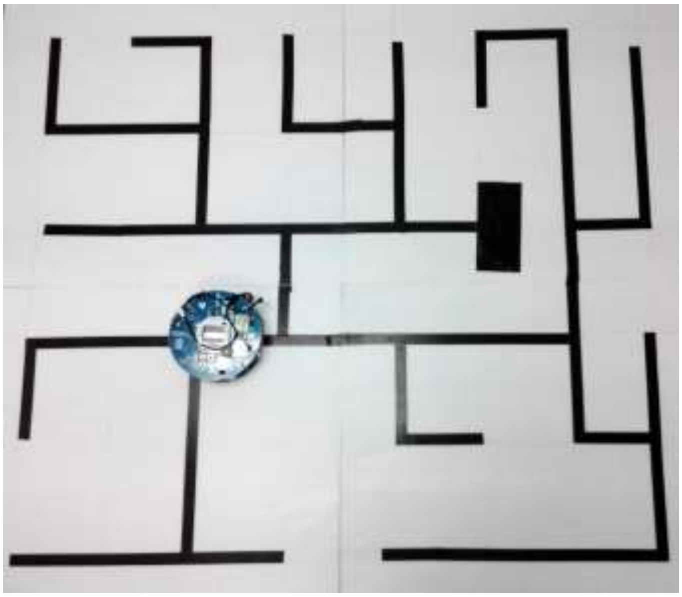
Figure 7.7 Example of the maze course

There are a couple of ways to solve this maze. The first one is to make sure the robot is able to explore the entire grid. By following every black line, the robot will eventually find the black space. The second way to look for the black space is to just wander around the maze randomly until the black space is found.

An alternative maze challenge using Artificial Intelligence is discussed in Chapter 8.

Chapter 8 Maze Puzzle

Chia Kim Seng



8.1 Introduction

Imagine that you are in a maze without a compass, a map, or any other tools that may help you get out from the maze. The good news is that you do have your brain and intelligence as that are always belong to you and non-transferable to other. Besides, based on previous records, your brain and related intelligence should be sufficient to help you to get out from the maze. However, a potential bad news is that you may not have that related intelligence to solve the puzzle in that moment.

As Bobby Unser once said – Success is where preparation and opportunity meet. This chapter aims to provide related intelligence that could be used to solve this kind of puzzle in the future. An example of assignment (Section 8.2) that was

designed to assess the understanding of Bachelor’s degree students on the use of Artificial Intelligence (AI) searching algorithm and production rule is provided as case study. The feedback from the assignment is presented in Section 7.3.

8.2 Example of Assignment

****

ARTIFICIAL INTELLIGENCE (BTIS 3043)

Assignment 5: Individual work (15%)

Semester 2 / Year 2014

Objective: To assess the understanding of artificial intelligence concept that related to real world operation and environment.

Instruction: Answer all questions

1. You are required to describe an AI concept to solve the following puzzle. Initially, a Robot is located as that illustrated in the Figure 1.0. The location of the Target, however, is unknown. The goal of the puzzle to “teach” the Robot to:

   i. explore or discover the Target (4%)
   ii. figure out the shortest path to reach the Target. (4%)

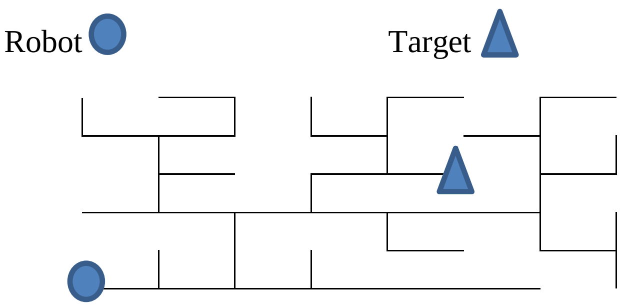
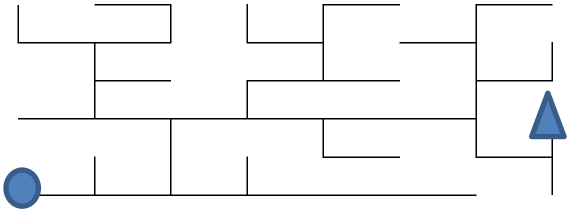


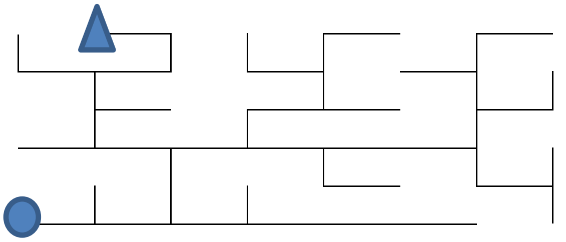
Figure 1.0

2. Upon the completion of Task 1, you are required to:

i. Test the AI concept to solve the following puzzle. (2%)



ii. Test the AI concept to solve the following puzzle. (2%)\



3. All the results and answers must be well presented and written in a short report. (3%)

Submission schedule:

Task	Due date
1. i.	Week 10
1. ii.	Week 11
2. i.	Week 12
2. ii.	Week 12
3	Week 13

Note: Zero mark will be given for late submission

****

8.2.1 Example Solution

1.i. Using left-hand rule: Make decision when a junction is detected as follows:

WHILE (Target has not been reached)
    IF can_turn_left? THEN turn_left
    ELSE IF can_go_straight? THEN go_straight
    ElSE IF can_turn_right? THEN turn_right
    ElSE U_turn

Symbol representation:

Actions	Symbol
Turn_left	L
Go_straight	S
Turn_right	R
U turn	U

Path of exploration:
Start->LULLLSULSLRULSLLURRLLULLSLRLLRULLRULSL->Target

1.ii. Using Production rule:

LUL = S	RUR = S	**RUL = U**	**LUR = U**	SUR=L
SUL = R	**SUS = U**	**LUS = R**	RUS = L	

State	**LEFT Hand Rule**
Path of exploration	Start->LULLLSULSLRULSLLURRLLULLSLRLLRULLRULSL->Target
First optimisation	Start->**LUL**LLSULSLRULSLLURRLLULLSLRLLRULLRULSL->Target Start->**S**LLSULSLRULSLLURRLLULLSLRLLRULLRULSL->Target
2	Start->SLL**SUL**SLRULSLLURRLLULLSLRLLRULLRULSL->Target Start->SLL**R**SLRULSLLURRLLULLSLRLLRULLRULSL->Target
3	Start->SLLRSL**RUL**SLLURRLLULLSLRLLRULLRULSL->Target Start->SLLRSL**U**SLLURRLLULLSLRLLRULLRULSL->Target
4	Start->SLLRSLUS**LLURR**LLULLSLRLLRULLRULSL->Target Start->SLLRSLUS**U**LLULLSLRLLRULLRULSL->Target
5	Start->SLLRSLU**SUL**LULLSLRLLRULLRULSL->Target Start->SLLRSLU**R**LULLSLRLLRULLRULSL->Target
6	Start->SLLRSLUR**LUL**LSLRLLRULLRULSL->Target Start->SLLRSLUR**S**LSLRLLRULLRULSL->Target
7	Start->SLLRS**LUR**SLSLRLLRULLRULSL->Target Start->SLLRS**U**SLSLRLLRULLRULSL->Target
8	Start->SLLR**SUS**LSLRLLRULLRULSL->Target Start->SLLR**U**LSLRLLRULLRULSL->Target
9	Start->SLL**RUL**SLRLLRULLRULSL->Target Start->SLL**U**SLRLLRULLRULSL->Target
10	Start->SL**LUS**LRLLRULLRULSL->Target Start->SL**R**LRLLRULLRULSL->Target
11	Start->SLRLRLL**RUL**LRULSL->Target Start->SLRLRLL**U**LRULSL->Target
12	Start->SLRLRL**LUL**RULSL->Target

	Start->**SLRLRLS**RULSL->Target
13	Start->**SLRLRLSRUL**SL->Target Start->**SLRLRLSU**SL->Target
14	Start->**SLRLRLSUS**L->Target Start->**SLRLRLU**L->Target
15	Start->**SLRLRLUL**->Target Start->**SLRLRS**->Target
Shortest path	**Start->SLRLRS->Target**

LUL = S **RUR = S** **RUL = U** **LUR = U** **SUR=L**
SUL = R SUS = U **LUS = R** **RUS = L**

Start-> SLRLRS->Target

State	**RIGHT Hand Rule (optimise from right hand)**
Path of exploration	Start-> SSSURURRRSRLURRRLRUSURRSRLURRSRULRURSRSRRS->Target
First optimisation	Start-> SSSURURRRSRLURRRLRUSURRSRLURRSRUL**RUR**SRSRRS->Target Start-> SSSURURRRSRLURRRLRUSURRSRLURRSRUL**S**SRSRRS->Target
2	Start-> SSSURURRRSRLURRRLRUSURRSRLURRS**RUL**SSRSRRS->Target Start-> SSSURURRRSRLURRRLRUSURRSRLURRS**U**SSRSRRS->Target
3	Start-> SSSURURRRSRLURRRLRUSURRSRLURR**SUS**SRSRRS->Target Start-> SSSURURRRSRLURRRLRUSURRSRLURR**U**SRSRRS->Target
4	Start-> SSSURURRRSRLURRRLRUSURRSRLUR**RUS**RSRRS->Target Start-> SSSURURRRSRLURRRLRUSURRSRLUR**L**RSRRS->Target
5	Start-> SSSURURRRSRLURRRLRUSURRS**RLURL**RSRRS->Target Start-> SSSURURRRSRLURRRLRUSURRS**U**RSRRS->Target
6	Start-> SSSURURRRSRLURRRLRUSURR**SUR**SRRS->Target Start-> SSSURURRRSRLURRRLRUSURR**L**SRRS->Target
7	Start-> SSSURURRRSRLURRRLRU**SUR**RLSRRS->Target Start-> SSSURURRRSRLURRRLRU**L**RLSRRS->Target
8	Start-> SSSURURRRSRLURR**RLRULRL**SRRS->Target Start-> SSSURURRRSRLURR**U**SRRS->Target
9	Start-> SSSURURRRSRLUR**RUS**RRS->Target Start-> SSSURURRRSRLUR**L**RRS->Target
10	Start-> SSSURURRRS**RLURL**RRS->Target Start-> SSSURURRRS**U**RRS->Target
11	Start-> SSSURURRR**SUR**RS->Target Start-> SSSURURRR**L**RS->Target
12	Start-> SSSU**RUR**RRLRS->Target Start-> SSSU**S**RRLRS->Target
13	Start-> SS**SUS**RRLRS->Target Start-> SS**U**RRLRS->Target
14	Start-> S**SUR**RLRS->Target Start-> S**L**RLRS->Target
Shortest path	**Start-> SLRLRS->Target**

LUL = S | **RUR = S** | RUL = U | **LUR = U**

SUR = L | SUL = R | **SUS = U**

LUS = R | **RUS = L**

State	**RIGHT Hand Rule**
Path of exploration	Start-> SSSURURRRSRLURRRLRUSURRSRLURRSRULRURSRSRRS->Target
First optimisation	Start-> SS**SUR**URRRSRLURRRLRUSURRSRLURRSRULRURSRSRRS->Target Start-> SS**L**URRRSRLURRRLRUSURRSRLURRSRULRURSRSRRS->Target
2	Start-> SS**LUR**RRSRLURRRLRUSURRSRLURRSRULRURSRSRRS->Target Start-> SS**U**RRSRLURRRLRUSURRSRLURRSRULRURSRSRRS->Target
3	Start-> S**SUR**RSRLURRRLRUSURRSRLURRSRULRURSRSRRS->Target Start-> S**L**RSRLURRRLRUSURRSRLURRSRULRURSRSRRS->Target
4	Start-> SLRSR**LUR**RRLRUSURRSRLURRSRULRURSRSRRS->Target Start-> SLRSR**U**RRLRUSURRSRLURRSRULRURSRSRRS->Target
5	Start-> SLRS**RUR**RLRUSURRSRLURRSRULRURSRSRRS->Target Start-> SLRS**S**RLRUSURRSRLURRSRULRURSRSRRS->Target
6	Start-> SLRSSRL**RUS**URRSRLURRSRULRURSRSRRS->Target Start-> SLRSSRL**L**URRSRLURRSRULRURSRSRRS->Target
7	Start-> SLRSSR**LLURR**SRLURRSRULRURSRSRRS->Target Start-> SLRSSR**U**SRLURRSRULRURSRSRRS->Target
8	Start-> SLRSS**RUS**RLURRSRULRURSRSRRS->Target Start-> SLRSS**L**RLURRSRULRURSRSRRS->Target
9	Start-> SLRSSLR**LUR**RSRULRURSRSRRS->Target Start-> SLRSSLR**U**RSRULRURSRSRRS->Target
10	Start-> SLRSSL**RUR**SRULRURSRSRRS->Target Start-> SLRSSL**S**SRULRURSRSRRS->Target
11	Start-> SLRSSLSS**RUL**RURSRSRRS->Target Start-> SLRSSLSS**U**RURSRSRRS->Target
12	Start-> SLRSSLS**SUR**URSRSRRS->Target Start-> SLRSSLS**L**URSRSRRS->Target
13	Start-> SLRSSLS**LUR**SRSRRS->Target Start-> SLRSSLS**U**SRSRRS->Target
14	Start-> SLRSSL**SUS**RSRRS->Target Start-> SLRSSL**U**RSRRS->Target
15	Start-> SLRS**SLURS**RRS->Target Start-> SLRS**U**RRS->Target
16	Start-> SLR**SUR**RS->Target Start-> SLR**L**RS->Target
Shortest path	**Start-> SLRLRS->Target**

8.3 Discussion & Learners' Feedbacks

The assignment that stated in Section 8.2 can be categorised as Problem Based Learning assignment.

Initially, a problem or challenge was explicitly explained to the students in written form.

Rome was not built in one day. Therefore, a **Submission Schedule** was used to ensure all the students are on the track of the assignment. Besides, students can use the schedule to find a balance between the given assignment and their other assignments from other subjects.

Once the students completed and submitted a task, a constructive discussion and feedback from the work done were conducted to rectify the direction of students before they proceed to the next tasks. This practice could reduce the deviation and enhance the confident level of the students to continue the coming tasks.

The following feedbacks are that extracted from the reports from my students in 2014.

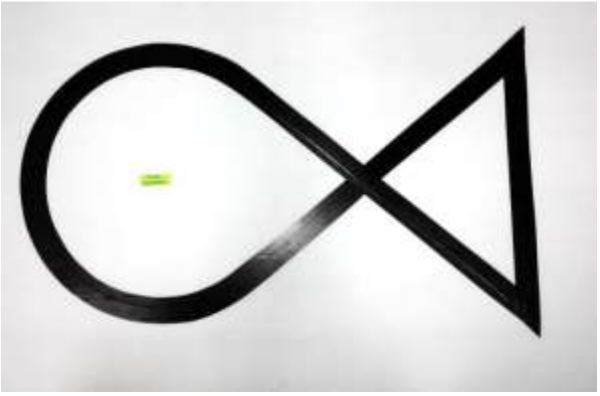
> "*I have learned a lot from this assignment. This is a great opportunity that we can learn how the robot working to explore and discover the target and then also learned how to find the shortest path.*"
>
> – Yong Jian Keong (BSE13-C2), 2014

> "*This is an unforgettable learning process for me ... I truly enjoyed the moment while doing this puzzle assignment…*"
>
> – Lim Jing Xiang (BSE13-C2), 2014

This feedback indicates that the proposed problem-based assignment helped students learn about AI through the experience of problem solving. Besides, the benefits of problem-based learning strategy such as helping students to develop flexible knowledge and effective problem solving skills were demonstrated at the end of the proposed assignment.

Chapter 9 C Programming Learning

Chia Kim Seng



9.1 Introduction

Learning computer programming language requires lots of self-motivation. This is because the best way to learn programming language is by practice. However, practice also implies that learners need to learn from their failure, i.e. trial-and-errors, that may demotivate the desire of learners to practise their skills and knowledge in the long run. Thus, a better approach is needed to motivate learners keep practising their programming knowledge.

To address this challenge, an assignment (Section 9.2) that involved the use of 3pi robot (Chapter 1) is designed to enrich the learning process of C programming course.

In order to ensure that the assignment is achievable and measureable within a limited time frame of a semester, an explicit instruction, resources and assessment scheme are vital.

Unlike the previous AI assignment that consists of a Submission Schedule, the proposed C programming assignment that used a line follower robot is assumed

to be interesting enough to inspire and motivate students to complete the assignment in their private period before a given due date.

Since we need to have a good knowledge about the hardware and software of the robot in order to complete a line follower robot, the challenge of designing the assignment is to ensure that sufficient information is given so that students can focus on using their C programming knowledge and skills to solve the given task – make the robot follows a given line.

In this chapter, firstly, the example of the designed assignment is presented (Section 9.2). Then, a holistic discussion is provided in Section 9.3.

9.2 Example of Assignment

****

C Programming (CSEG 1113)
Assignment 3: (16.5%)

Group assignment: At most 3 people in one group
Semester 2 / Year 2014

Line Following Robot Assignment

Objective:
To assess the ability to use of Array and Function for problem solving.

Instruction:
You are required to use C language and creativity to make a robot move along the given line. Besides using control functions (e.g. if else, while, for and switch), you may need to call three internal functions to control the robot, i.e. **updateIR()**, **motorsWrite()**, and **motorStop()**. The descriptions of these functions can be found in the following pages.

There are four main tasks you need to handle in this project as stated in the playing field, i.e., move along a straight line, move along a curve, move along a square, and move along a triangle.

Assessment method:

Criteria	Mark
Report: Code + description	5%
Robot performance	2.5%
Team work	2%
Presentation and Q&A	7%
Total	**16.5%**

For report writing and oral presentation, you are expected to present at least the following items:

1. How does your program overcome the four main tasks in this project?
2. The most challenging and interesting tasks in this project.

The Description of internal functions

Pololu3pi.readLine( )

Syntax:
unsigned int Pololu3pi::readLine(unsigned int **sensorValues*, unsigned char ***readMode*** = IR_EMITTERS_ON, unsigned char ***whiteLine*** = 0)

Note: Need to include head file of pololu/3pi.h

Example:

```
unsigned int sensors[5];

Pololu3pi.readLine(sensors, IR_EMITTERS_ON);
```

Description:
sensors[0] will return the a value between 0 and 1000 that indicates the position of the first IR sensor from the left hand side. Value of 0 indicates 100% white while value of 1000 indicates 100% black. Users can control the motors based on the information of these values from the five IR sensors.

OrangutanMotors::setSpeeds( )
Syntax:
static void OrangutanMotors::setSpeeds(int ***m1Speed***, int ***m2Speed***)

This method will set the speeds and directions of motors 1 and 2. First argument is for motor in the left side; while the second argument is for the motor in the right side. The value is between -255 to 255.

Example:

```
OrangutanMotors::setSpeeds(70, 70); // both wheels with same speed forward
OrangutanMotors::setSpeeds(0, 70);  // left motor is stop; right motor is moving,
                                    //then robot turns left.
OrangutanMotors::setSpeeds(70, 0); // left motor is moving; right motor is stop;
                                    //then robot turns right
```

Description:

Even though the maximum speed is up to magnitude of 255, it is advised that to start your robot with speed between 50 and 80. When you can make your robot moves smoothly with that speed, then you are encouraged to try slightly higher speed e.g. 90-150. Normally, the difficulty to make a robot moves smoothly will increase proportional to the speed of movement.

Example Framework

```
/*
 * Programmed by: Kim Seng Chia
 * Date: May 2014
 */

#include <Pololu3pi.h>
#include <PololuQTRSensors.h>
#include <OrangutanMotors.h>
#include <OrangutanAnalog.h>
#include <OrangutanLEDs.h>
#include <OrangutanLCD.h>
#include <OrangutanPushbuttons.h>
#include <OrangutanBuzzer.h>

Pololu3pi robot;
unsigned int sensors[5]; // an array to hold sensor values

// This include file allows data to be stored in program space.  The
// ATmega168 has 16k of program space compared to 1k of RAM, so large
// pieces of static data should be stored in program space.

void setup(){
  unsigned int counter; // used as a simple timer

   // This must be called at the beginning of 3pi code, to set up the
  // sensors.  We use a value of 2000 for the timeout, which
  // corresponds to 2000*0.4 us = 0.8 ms on our 20 MHz processor.
```

```
  robot.init(2000);

  while (!OrangutanPushbuttons::isPressed(BUTTON_B)){
   int bat = read_battery_millivolts();
   clear();
   print_long(bat);
   print("mV");
   lcd_goto_xy(0,1);
   print("Press B");
   delay(100);
  }

   // Always wait for the button to be released so that 3pi doesn't
  // start moving until your hand is away from it.

  OrangutanPushbuttons::waitForRelease(BUTTON_B);
  delay(1000);

   // Auto-calibration: turn right and left while calibrating the sensors.
  for (counter=0; counter<80; counter++){
   if (counter < 20 || counter >= 60)
     OrangutanMotors::setSpeeds(40, -40);
   else
     OrangutanMotors::setSpeeds(-40, 40);

   // This function records a set of sensor readings and keeps
   // track of the minimum and maximum values encountered.  The
   // IR_EMITTERS_ON argument means that the IR LEDs will be
   // turned on during the reading, which is usually what you want.

   robot.calibrateLineSensors(IR_EMITTERS_ON);

   // Since our counter runs to 80, the total delay will be
   // 80*20 = 1600 ms.
   delay(20);
  }
  OrangutanMotors::setSpeeds(0, 0);

   while (!OrangutanPushbuttons::isPressed(BUTTON_B))
       delay(100);
  OrangutanPushbuttons::waitForRelease(BUTTON_B);
       delay(5);
}

 // The main function.  This function is repeatedly called by the Arduino framework.
void loop(){
  robot.readLine(sensors, IR_EMITTERS_ON);
  if(sensors[1] < 0 && sensors[2] < 0 && sensors[3] < 0)
  OrangutanMotors::setSpeeds(0, 0);
 else
  OrangutanMotors::setSpeeds(20, 20);
}
```

Main Reference:
Getting started with 3pi Robot (Chapter 1)
Extra reference:
Pololu 3pi Robot User's Guide
****

9.3 Discussion and learners' Feedback

In order to save the time needed to explore the way used to interface the given robot (i.e. Pololu 3pi Robot), the first draft of Chapter 1 of this handbook was given to the students to assist them getting started with the 3pi Robot. This was taken around 4 hours (private time) without supervision for my students to successfully interface the robot with their laptop, and also to make the LED of the robot blinks. This experience helped them to have a better understanding about their programming skills via observing tangible outcomes.

After that, my students scheduled another afternoon to take the 3pi robots from me for them to continue their assignment in private time. Again, this is another 4 hours self-learning session without my supervision. Nonetheless, a short conversation was conducted immediately after they returned the robots to me. This is because I would like to understand their progress.

It is worthy to mention that the robot can follow the given line after the 8 hours self-learning was spent. This progress was good. Thus, I encouraged them to take a video of their outcome and share the video with their peer via social media e.g. Facebook, as a record of their learning journey.

Nonetheless, their program was only able to overcome two of the four tasks, i.e. "move along a straight line" and "move along a curve". Another two tasks of "move along a square" and "move along a triangle" were still remaining for them to solve using their C programming knowledge and skills.

Immediately, my students scheduled another afternoon to take the robots and to continue their assignment. Unfortunately, no obvious progress was achieved after another 4 hours self-learning session. This should not be an unexpected outcome based on my experience. This is because sometimes we need to think out of the box in order to solve a programming puzzle. In a short conversation with my students about their progress, I shared some ideas to inspire them to solve the challenge verbally.

Next day, with a great desire to solve the puzzle using some ideas that we had discussed, ultimately, my students solved the challenge by their own effects with countless trial-and-errors. This was another 4 hours self-learning session.

Even though the four main tasks of the assignment had been completed, my students spent another 4 hours to optimise the performance of the robot using their programming skills. Besides, the robot with optimal performance was captured and shared with their peers via social media.

In short, there were at least 20 traceable self-organised self-learning hours spent by my students for this assignment, excluding the time spent for oral and written presentations preparation.

My observation was in agreement with the feedback written in my students' report as that stated below:

> "*During these two weeks, whether we have failed, we didn't try to give up but just keep trying. Initially, the robot just only keep going on straight line and does not turn left or right. The robot also cannot stop when at the white lattice and cannot identify the direction when at the corner. Throughout our continuous efforts and improve, finally, the robot can run properly at this type of road.*"
>
> – Dee Kai Cher, Wong Wei Jien et. al. (DEE13-C), 2014

The use of robots in learning and teaching of C programming course is a promising way to inspire and motivate students to be more independent learners. The idea of the proposed assignment is in agreement with the philosophy of problem-based learning, particularly, in helping students to develop self-directed learning, and intrinsic motivation.

Appendix A

boards.txt

```
##############################################################
orangutan48pgm.name=Pololu Baby Orangutan B-48 via Programmer
orangutan48pgm.upload.using=avrispv2
orangutan48pgm.upload.maximum_size=4096
orangutan48pgm.build.mcu=atmega48
orangutan48pgm.build.f_cpu=20000000L
orangutan48pgm.build.core=arduino
orangutan48pgm.build.variant=standard
##############################################################
orangutan168pgm.name=Pololu Orangutan or 3pi robot w/ ATmega168 via Programmer
orangutan168pgm.upload.using=avrispv2
orangutan168pgm.upload.maximum_size=16384
orangutan168pgm.build.mcu=atmega168
orangutan168pgm.build.f_cpu=20000000L
orangutan168pgm.build.core=arduino
orangutan168pgm.build.variant=standard
##############################################################
orangutan328pgm.name=Pololu Orangutan or 3pi robot w/ ATmega328P via Programmer
orangutan328pgm.upload.using=avrispv2
orangutan328pgm.upload.maximum_size=32768
orangutan328pgm.build.mcu=atmega328p
orangutan328pgm.build.f_cpu=20000000L
orangutan328pgm.build.core=arduino
orangutan328pgm.build.variant=standard
```

programmers.txt

```
avrispv2.name=AVR ISP v2
avrispv2.communication=serial
avrispv2.protocol=avrispv2
```

Appendix B

Eng Pei Chee

```
module lineFollowing(
	clk,
	reset,
	u,
	m1_pwm,
	m2_pwm,
	m1_speed,
	m2_speed,
	led
);
//==========================================================
//  PARAMETER declarations
//==========================================================
parameter [7:0] speed_fast =100, speed_medium = 75, speed_slow = 0;
//==========================================================
//  PORT declarations
//==========================================================
input				clk;
input				reset;
//for verification purpose
output [7:0] led;
//sensor input
input [5:1] u;
wire [5:1] sensor = {u[1],u[2],u[3],u[4],u[5]};
//motor output
output m1_pwm;
output m2_pwm;
output [7:0] m1_speed;
output [7:0] m2_speed;
//==========================================================
//  INTERNAL REG & WIRE
//==========================================================
reg m1_pwm;
reg m2_pwm;
reg [7:0] m1_speed;
reg [7:0] m2_speed;
//==========================================================
//  STATE MACHINE
//==========================================================
(* syn_encoding = "safe" *)  reg [2:0] y, Y;
```

```
parameter forward = 0,
                    left = 1,
                    right = 2,
                    stop = 3,
                    junction_R = 4,
                    junction_L = 5,
                    left_slow =  6,
                    right_slow =  7;
//next state transition
always @ (u, y)
    case(y)
        forward:
            casex(sensor)
                5'b00000: Y = stop;
                5'b01100: Y = left_slow;
                5'b11000: Y = left;
                5'b10000: Y = left;
                5'b00001: Y = right;
                5'b00011: Y = right;
                5'b00110: Y = right_slow;

                5'b01111: Y = junction_R;
                5'b11110: Y = junction_L;

                default: Y = forward;
            endcase
        left:
            casex(sensor)
                5'b01110: Y = forward;
                5'b01100: Y = left_slow;
                5'b00001: Y = right;
                5'b00011: Y = right;
                5'b00110: Y = right_slow;
                5'b01111: Y = junction_R;
                5'b11110: Y = junction_L;

                default: Y = left;
            endcase
        left_slow:
            casex(sensor)
                5'b01110: Y = forward;
                5'b11000: Y = left;
                5'b10000: Y = left;
                5'b00001: Y = right;
                5'b00011: Y = right;
                5'b00110: Y = right_slow;
                5'b01111: Y = junction_R;
                5'b11110: Y = junction_L;

                default: Y = left_slow;
            endcase
```

```
right:
        casex(sensor)
                5'b01110: Y = forward;
                5'b01100: Y = left_slow;
                5'b11000: Y = left;
                5'b10000: Y = left;

                5'b00110: Y = right_slow;

                5'b01111: Y = junction_R;
                5'b11110: Y = junction_L;

                default: Y = right;
        endcase
right_slow:
        casex(sensor)
                5'b01110: Y = forward;
                5'b01100: Y = left_slow;
                5'b11000: Y = left;
                5'b10000: Y = left;

                5'b00001: Y = right;
                5'b00011: Y = right;

                5'b01111: Y = junction_R;
                5'b11110: Y = junction_L;

                default: Y = right_slow;
        endcase
stop:



        casex(sensor)
                5'b01110: Y = forward;

                5'b01100: Y = left_slow;
                5'b11000: Y = left;
                5'b10000: Y = left;

                5'b00001: Y = right;
                5'b00011: Y = right;
                5'b00110: Y = right_slow;

                default: Y = stop;
        endcase
junction_R:
        casex(sensor)
                5'b01110: Y = forward;
                5'b00000: Y = right;
                default: Y = junction_R;
        endcase
```

```
            junction_L:
                        casex(sensor)
                                    5'b01110: Y = forward;
                                    5'b00000: Y = left;
                                    default: Y = junction_L;
                        endcase
            default: Y = forward;
      endcase

//sequential block
always @ (posedge clk)
      if(reset == 0) y <= stop;
      else y <= Y;
//output
always @ (Y)
      case(Y)
            forward:
                        begin
                        m1_pwm <= 1;
                        m2_pwm <= 1;
                        m1_speed <= speed_fast;
                        m2_speed <= speed_fast+15;
                        end
            left:
                        begin
                        m1_pwm <= 1;
                        m2_pwm <= 1;
                        m1_speed <= speed_slow;//
                        m2_speed <= speed_medium;
                        end
            right:
                        begin
                        m1_pwm <= 1;
                        m2_pwm <= 1;
                        m1_speed <= speed_medium;//
                        m2_speed <= speed_slow;
                        end
            junction_L:
                        begin
                        m1_pwm <= 1;
                        m2_pwm <= 1;
                        m1_speed <= speed_medium-10;
                        m2_speed <= speed_medium-10;
                        end
            junction_R:
                        begin
                        m1_pwm <= 1;
                        m2_pwm <= 1;
                        m1_speed <= speed_medium-10;
                        m2_speed <= speed_medium-10;
                        end
```

```
            stop:
                begin
                m1_pwm <= 0;
                m2_pwm <= 0;
                m1_speed <= 0;
                m2_speed <= 0;
                end
            left_slow:
                begin
                m1_pwm <= 1;
                m2_pwm <= 1;
                m1_speed <= speed_medium-30;//
                m2_speed <= speed_medium;
                end
            right_slow:
                begin
                m1_pwm <= 1;
                m2_pwm <= 1;
                m1_speed <= speed_medium;//
                m2_speed <= speed_medium-30;
                end
            default:
                begin
                m1_pwm <= 1;
                m2_pwm <= 1;
                m1_speed <= speed_medium;
                m2_speed <= speed_medium;
                end
        endcase

    //for verification purposes
    assign led[0] = (y==forward);
    assign led[1] = (y==left);
    assign led[2] = (y==right);
    assign led[3] = (y==stop);
    assign led[4] = (y==junction_R);
    assign led[5] = (y==junction_L);
    assign led[6] = (y==left_slow);
    assign led[7] = (y==right_slow);

    endmodule
```

Appendix C

Eng Pei Chee

```
module DE0_Nano_FSM(
	CLOCK_50,  // CLOCK
	LED,    // LED
	KEY,     // KEY
	// GPIO_0, GPIO_0 connect to GPIO Default
	GPIO_0,
	GPIO_0_IN,
);

//=======================================================
//  PORT declarations
//=======================================================
input                               CLOCK_50;   // CLOCK
output              [7:0]           LED;     // LED
input               [1:0]           KEY;     // KEY
// GPIO_0, GPIO_0 connect to GPIO Default
inout               [33:0]          GPIO_0;
input               [1:0]           GPIO_0_IN;

//=======================================================
//  INTERNAL REG & WIRE
//=======================================================
wire pwm1;
wire pwm2;
wire [7:0] m1_speed;
wire [7:0] m2_speed;

wire clk_10MHz;
wire clk_25MHz;
wire clk_100MHz;

//=======================================================
//  MODULE INSTANTIATION
//=======================================================
//motor direction: 0-forward, 1-reverse
assign GPIO_0[0] = 1'b0;
assign GPIO_0[3] = 1'b0;
```

```
//pll
pll pll_this(
	.inclk0(CLOCK_50),
	.c0(clk_10MHz),
);
//pwm signal
pwm pwm_m1(
	.clock(clk_10MHz),
	.stepCountBit(m1_speed),
	.pwm(GPIO_0[1]),   //m1
	.pwm_value(pwm1),
	.reset(KEY[0])
);
pwm pwm_m2(
	.clock(clk_10MHz),
	.stepCountBit(m2_speed),
	.pwm(GPIO_0[5]),   //m2
	.pwm_value(pwm2),
	.reset(KEY[0])
);

lineFollowing lineFollowing_this(
	.clk(clk_10MHz),
	.reset(KEY[0]),
	.u({GPIO_0[25],GPIO_0[27],GPIO_0[29],GPIO_0[31],GPIO_0[33]}),
	.m1_pwm(pwm1),
	.m2_pwm(pwm2),
	.m1_speed(m1_speed),
	.m2_speed(m2_speed),
	.led(LED),
);

endmodule
```

Build a Line Follower Robot

- A User-Friendly Guide

Theoretical knowledge and practical skills are equally important for engineering learners. There is a challenge for engineering lecturers to convey engineering knowledge in such ways that students will be capable of using the knowledge to solve “real-life” engineering problems in the nearly future.

This handbook aims to provide an alternative for learners to learn and practice their engineering knowledge by building their own line follower robots. Besides, lecturers can use this handbook to enrich their teaching content using practical examples.

We use open-source electronics platform based on easy-to-use hardware and software to demonstrate practical examples so that learners can carry out similar projects with minimum costs.

While completing this handbook, you are expected to have a better understand about motor control, IR sensors, and C programming. Using these knowledge and experience, you are capable of building various projects.

Inspiring Great Learning
南方大學 南方之強
SOUTHERN UNIVERSITY COLLEGE
南方大學學院

eISBN 978-983-2453-46-8
9 789832 453468